

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

A HEURISTIC SEARCH METHOD
OF SELECTING RANGE-RANGE SITES
FOR HYDROGRAPHIC SURVEYS

by

Arnold F. Steed

September, 1991

Thesis Advisor:
Co-Advisor:

Neil C. Rowe
Everett Carter

Approved for public release; distribution is unlimited

T258738

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b. OFFICE SYMBOL (If applicable) 35		7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000			7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			Program Element No.	Project No.	Task No.
11. TITLE (Include Security Classification) A HEURISTIC SEARCH METHOD OF SELECTING RANGE-RANGE SITES FOR HYDROGRAPHIC SURVEYS					
12. PERSONAL AUTHOR(S) Steed, Arnold F.					
13a. TYPE OF REPORT Master's Thesis		13b. TIME COVERED From To		14. DATE OF REPORT (year, month, day) September, 1991	
15. PAGE COUNT 123					
16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.					
17. COSATI CODES			18. SUBJECT TERMS (continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUBGROUP	Hydrography, navigation, heuristic search, artificial intelligence		
19. ABSTRACT (continue on reverse if necessary and identify by block number) One of the costliest aspects of many hydrographic surveys is establishing and occupying the navigation control stations. As budget cuts force agencies to conduct their surveys more efficiently, minimizing the cost of these control networks will be of primary importance. Because it has the ability to process numerical information faster than a human, a computer could be used to assist the survey planner in selecting optimal shore sites, yet little work has actually been done in this area. This thesis examines the possibility of using Artificial Intelligence (AI) to assist the survey planner in selecting navigation control sites. A search program is presented which uses a number of heuristics to select sites and guide the search for an optimal solution. The program was tested in several actual and idealized survey situations, and the results of these tests indicate that the heuristic search approach has the potential of surpassing a human expert in the selection of an optimal set of sites.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS REPORT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL Neil C. Rowe			22b. TELEPHONE (Include Area code) (408) 646-2462		22c. OFFICE SYMBOL CS/Rp

Approved for public release; distribution is unlimited.

A Heuristic Search Method
of Selecting Range-Range Sites
for Hydrographic Surveys

by

Arnold F. Steed
Physical Scientist, Naval Oceanographic Office
B.S., University of Texas , 1987

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN HYDROGRAPHIC SCIENCE

from the

ABSTRACT

One of the costliest aspects of many hydrographic surveys is establishing and occupying the navigation control stations. As budget cuts force agencies to conduct their surveys more efficiently, minimizing the cost of these control networks will be of primary importance. Because it has the ability to process numerical information faster than a human, a computer could be used to assist the survey planner in selecting optimal shore sites, yet little work has actually been done in this area.

This thesis examines the possibility of using Artificial Intelligence (AI) to assist the survey planner in selecting navigation control sites. A search program is presented which uses a number of heuristics to select sites and guide the search for an optimal solution. The program was tested in several actual and idealized survey situations, and the results of these tests indicate that the heuristic search approach has the potential of surpassing a human expert in the selection of an optimal set of sites.

1/10/20
567863
C.1

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	HYDROGRAPHIC SURVEY PLANNING	1
B.	OPTIMIZING SITE LOCATIONS	2
C.	OVERVIEW OF THESIS	3
II.	DETAILED PROBLEM STATEMENT	5
A.	HORIZONTAL CONTROL FOR HYDROGRAPHIC SURVEYS . .	5
B.	PRACTICAL CONSTRAINTS FOR SITE SELECTION . . .	7
C.	EXISTING PROGRAMS	8
D.	STRATEGIES FOR OPTIMIZATION	9
1.	Simulated Annealing	9
2.	Search Algorithms	9
E.	TWO REPRESENTATIONS OF THE PROBLEM	10
1.	Grid Approximation	10
2.	Polygon Approximation	11
F.	ASSUMPTIONS MADE IN THE PROGRAM	11
1.	The Coordinate System	11
2.	The Search Space	12
3.	Positioning System	13
G.	SUMMARY	14
III.	DESCRIPTION OF PROGRAM	15
A.	OVERVIEW OF SOFTWARE AND HARDWARE	15
B.	SEARCH HEURISTICS DEVELOPED	15

C.	MAJOR COMPONENTS OF PROGRAM	16
1.	Data Structures	16
2.	Input and Output	18
3.	The Search Algorithm	19
D.	THE GOAL STATE	20
E.	THE COST FUNCTION	20
1.	Total Cost	20
2.	Actual Network Cost	21
3.	Artificial Geometric Cost	21
F.	THE EVALUATION FUNCTION	22
G.	THE SUCCESSOR FUNCTION	25
H.	SUMMARY	26
IV.	SUCCESSOR FUNCTION HEURISTICS	27
A.	THE BASIC STRATEGY	27
B.	CHOOSING THE INITIAL SITE	28
C.	CHOOSING SUBSEQUENT SITES	29
D.	APPLYING THE HEURISTICS	32
E.	UPDATING THE REMAINING SURVEY AREA	34
F.	THE SEARCH SPACE	37
G.	SUMMARY	39
V.	RESULTS OF TESTING	40
A.	THE TEST DATA SETS	40
B.	THE SUCCESSOR HEURISTICS	41
1.	The Center of Polygon Function	41
2.	The Optimal Solution	43
3.	Heuristics Used in Optimal Solutions	44

C.	THE EVALUATION FUNCTION	46
D.	SPACE AND TIME USAGE	52
E.	THE SECONDARY DATA SETS	52
F.	SUMMARY	55
VI.	CONCLUSIONS	57
A.	GENERAL CONCLUSIONS	57
B.	ADDITIONAL RELATED WORK	58
C.	MODIFICATIONS TO THE HEURISTIC SEARCH PROGRAM .	58
1.	Shifting Existing Sites	58
2.	Fixed Search Space	59
3.	Grid Approximation	59
D.	SUMMARY	60
	APPENDIX A - THE PRIMARY DATA SETS	61
	APPENDIX B - THE SECONDARY DATA SETS	80
	APPENDIX C - PROGRAM SOURCE CODE	86
	LIST OF REFERENCES	112
	INITIAL DISTRIBUTION LIST	113

ACKNOWLEDGMENTS

Many thanks go to Neil Rowe, Everett Carter, and Bob Marks for their programming assistance. I also owe a special debt of gratitude to Kurt Schnebele for his ideas and criticism. Without their help this project would have been a lot more painful than it was.

But most of all I must thank my wife, Robbie, and my daughters, Laura and Cathryn. They have endured months of my moodiness, insomnia, and neglect with unfailing love and patience. Their support has made this thesis possible.

I. INTRODUCTION

A. HYDROGRAPHIC SURVEY PLANNING

Hydrographic surveys involve collecting many different types of data, including position fixes, sonar soundings, tidal heights, bottom samples, and the temperature, salinity, and pressure of the sea water. Coordinating all of these activities requires careful planning.

In order to accurately record a sounding on a nautical chart, two things must be known: the depth of the water (a vertical measurement) and the position of the sounding (a horizontal measurement). The position is usually expressed either in latitude and longitude or in northing and easting. Most methods for establishing the horizontal position require a number of fixed navigation control sites at known positions on shore. Although satellite positioning via GPS promises to revolutionize navigation and precise positioning of the survey vessel, application of this new technology is not proceeding as quickly as predicted. As a result, the need for shore-based navigation control sites will continue for some time.

One of the costliest aspects in many hydrographic surveys is establishing and occupying these sites. The person responsible for selecting appropriate sites is the survey planner. He must select sites to meet the requirements of the

survey, or provide sufficient positional accuracy at all points within the survey limits. The cost of each site depends on a number of factors, and is explained in greater detail in Chapter II. The survey planner should find a set of sites that meet the survey requirements at a minimum cost. At present, however, few tools exist to help find the optimal site locations.

B. OPTIMIZING SITE LOCATIONS

Currently, survey planning is highly subjective and governed by a few very loose rules of thumb, such as "try to place three sites in an equilateral triangle around the survey area." It is often easy to overlook simple solutions, or to stop as soon as a solution is found without trying to find a better one. There is also the temptation to use a particular configuration because it worked in the last survey. As budget cuts force agencies to conduct surveys more efficiently--to do more with less--a more systematic method of selecting sites will be needed.

Computers have several advantages in this regard. They tirelessly carry out repetitive and monotonous tasks, they can be more objective in their analysis, they are less likely to pre-judge a situation, and they can examine more possibilities than a human survey planner. All of this gives a computer the potential of finding an optimal or near-optimal solution.

Our program is similar to an expert system, in that it uses heuristics to make decisions such as where to locate sites and which set of sites is more useful. Unlike an expert system, however, our program surpasses what a single human expert can do. It examines more combinations of sites than a human can practically examine, and therefore has the potential of finding solutions that would not occur to a human expert.

C. OVERVIEW OF THESIS

The objective of this thesis is to study the viability of using a computerized expert system to recommend navigation control sites for a hydrographic survey. Our study focuses on three main objectives:

1. To develop a set of site selection heuristics which would ensure that the optimal solution is included in the search space.
2. To develop a set of cost estimation heuristics to efficiently direct the search.
3. To develop a set of polygon functions needed for the application of the search heuristics.

Chapter II begins with an explanation of the horizontal control required for a hydrographic survey, and continues with a discussion of some of the practical constraints limiting the selection of potential sites. The chapter then discusses how the problem was represented, as well as the assumptions made by the program.

Chapter III discusses the program in detail, with the primary focus being on the cost and evaluation functions as well as the geometric functions used by them. The successor function is also discussed briefly.

Chapter IV is devoted to an in-depth discussion of the successor function. The heuristics used by this function are discussed, as well as the geometric functions required to apply these heuristics.

Chapter V presents the results of our testing, and compares our heuristic evaluation function with a true A* evaluation function. A comparison is made between the various successor heuristics used. Finally, results of testing some of our geometric algorithms are presented.

In Chapter VI, we discuss the conclusions we have drawn from the test results, and further work that could be done on this project.

II. DETAILED PROBLEM STATEMENT

A. HORIZONTAL CONTROL FOR HYDROGRAPHIC SURVEYS

The International Hydrographic Organization (IHO), which publishes accuracy standards for hydrographic surveys, requires that the true position of a sounding be within 1.5 mm of its plotted position at the scale of the survey with a 95% probability (IHO, 1987).

This precise positioning is usually accomplished via an electronic positioning system such as range-range, which requires that radio positioning transmitters and receivers be set up at shore sites and on the survey vessel. The distance between the vessel and a shore site is determined by measuring either the travel time or the phase difference between a transmitted and received signal. By comparing the distances from two or more shore sites, an estimate of the vessel's position may be obtained.

The accuracy of a position obtained in this manner depends not only upon the accuracy of the distance measurement, but also upon the relative positions of the shore sites with respect to the vessel (Laurilla, 1976). For a two-site position fix, the angle of intersection (β) is defined as the angle at the vessel subtended by the shore sites. The accuracy of a position fix is given by

$$D_{rms} = \frac{\sqrt{2}\sigma}{\sin\beta} \quad (1)$$

where σ is the standard deviation of the range equipment. The ideal intersection angle is 90° , and the angle should generally be kept between 30° and 150° in the entire survey area (Umbach, 1976). It can be proven geometrically that this range of angles will occur as shown in Figure 1 (Wells and Hart, 1915). The area outside the circles contains intersection angles less than 30° , and the overlap area contains angles greater than 150° .

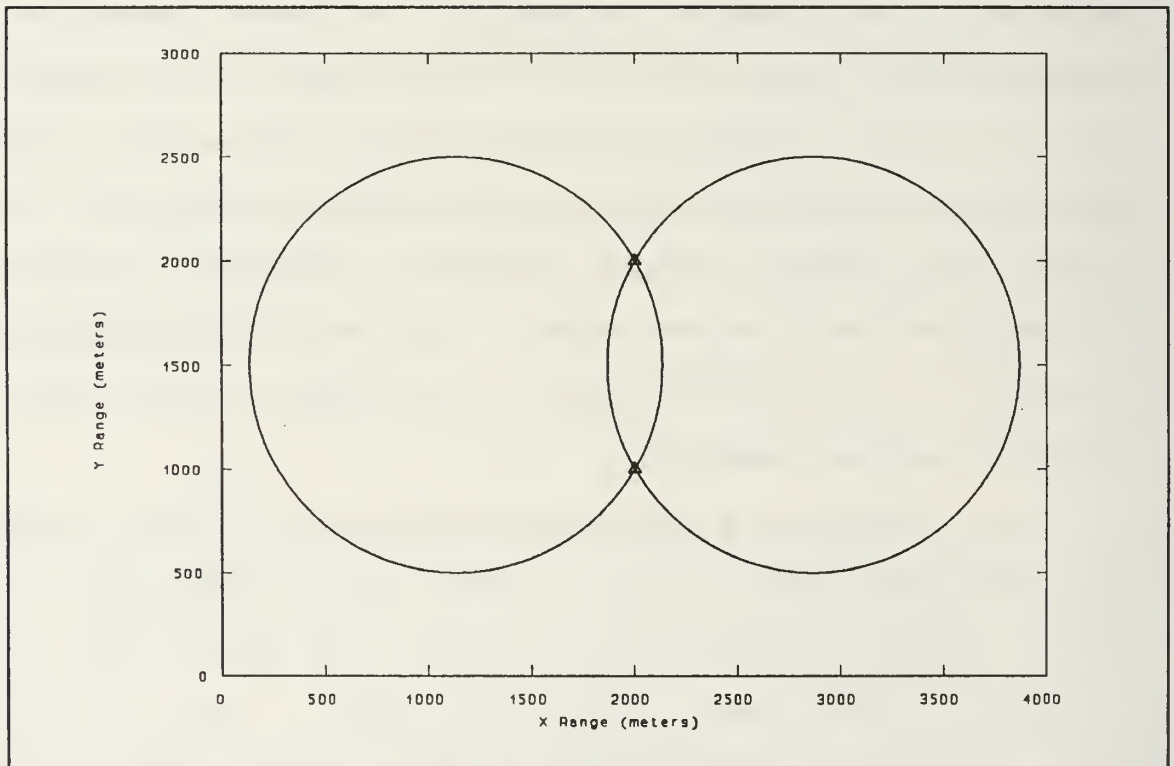


Figure 1. Geometric Coverage of Two Range Stations. The triangles indicate the positions of the stations.

The challenge to the survey planner is to select navigation sites such that every point in the survey area has

a good angle of intersection from at least two sites, while minimizing the cost of constructing and maintaining the site network.

B. PRACTICAL CONSTRAINTS FOR SITE SELECTION

When selecting prospective shore sites for a survey, the survey planner must consider more than just coverage of the survey area. Factors such as site security, ease of access and resupply, type of equipment needed, suitability of terrain, and existing geodetic control are also important (Clark, 1988). These factors must be weighed against each other. The result of this is that different sites may have different costs associated with them, and simply minimizing the number of sites may not be sufficient to minimize the overall cost of the network.

The cost of an individual site can be divided into two parts: the cost of establishing the site and the cost of maintaining the site. The ratio of the cost of establishing a site with the cost of maintaining a site depends not only on geographic location, but also on the type of equipment used and the duration of the survey.

For example, if a short survey is conducted using Minirangers (a short-range positioning system), the site maintenance only consists of visiting the site every two or three days to change the battery, if the site is secure. The cost of establishing the sites is then a significant part of

the overall cost. On the other hand, a survey using ARGO (a medium-range system) requires allocating personnel to man the sites during survey operations, and the cost of establishing the site will be almost negligible compared with the maintenance cost if operations will extend over a long period of time.

C. EXISTING PROGRAMS

We have been unable to find any previous use of computer programs to select optimal sites. Most software currently used to assist the survey planner is similar to that used by the Monterey Bay Aquarium Research Institute (MBARI). In this system, a list of navigation sites are entered into a file. The user can interactively select any combination of prospective sites from this list, and the software will draw contours of expected position accuracy for that set of sites. These accuracy values are obtained by a least squares computation based on the accuracy and positions of the shore sites. The user can then tell by inspection if that particular network meets the requirements of the survey (Meridian Ocean Systems, 1989). Final site selection, and hence any optimization, is done by the user through trial and error.

D. STRATEGIES FOR OPTIMIZATION

1. Simulated Annealing

Our optimization problem can be considered a minimization of a function of many variables; namely, the number of sites and position of each site. One method of finding this minimum is a technique known as simulated annealing (Kirkpatrick, Gelatt, and Vecchi, 1983). Under the proper conditions, this technique is virtually guaranteed to find the minimum, or optimal, solution. Unfortunately, our problem does not seem to lend itself to this type of solution.

Simulated annealing requires starting from a solution state, and assumes a method of obtaining a new solution state from an existing solution state. All possible solution states must be obtainable in this manner, from the initial state. This is difficult to accomplish in our problem. Obtaining an initial solution state (i.e., a set of navigation sites that satisfy the survey requirements) is a non-trivial problem in itself, and there is no simple or obvious method of obtaining a new solution state from an existing one--a change in the number or position of the existing sites may result in a state that is not a solution at all.

2. Search Algorithms

A number of search algorithms have been developed to find optimal solutions which do not require starting from a

solution state. One of the most useful of these algorithms is A* search.

This type of search requires a cost function to compute the cost of a state, an evaluation function to estimate the cost of going from a state to a solution state, and a successor function to define new states from an existing state. At each level of the search, the program picks the state with the lowest sum of cost and evaluation functions and generates the successors of that state. This process continues until a solution is found. Under the right conditions, the first solution found is guaranteed to be the optimal solution (Rowe, 1988).

We have therefore attempted to create an A* program for the site selection problem. We have also developed a number of heuristics for evaluating states and defining successor states. Although our final program does not meet all of the requirements for a true A* search, the results of our testing indicate that it produces optimal or near-optimal solutions for most of the cases studied.

E. TWO REPRESENTATIONS OF THE PROBLEM

1. Grid Approximation

One approach that we considered was to approximate the survey area by a set of equally spaced grid points. Testing the coverage of a site network would be accomplished by computing the position error of each grid point via a least-

squares computation. If the position error of a point is within the accuracy requirements of the survey, the point is covered; otherwise, it is not. The advantage of this method is that the positional accuracy is computed directly, making it very easy to accommodate different equipment accuracies and different survey requirements, including multiple ranges. The disadvantage is that this representation would make it very difficult to apply some of the heuristics we developed.

2. Polygon Approximation

The approach we finally adopted involves approximating the site coverage with polygons, as illustrated in Figure 2. In this approach, the coverage of the survey area is tested by finding the intersection of the coverage polygons with the survey area polygons. The advantage of this technique is that the survey area can be accurately represented, and the heuristics we developed for site selection can be easily implemented. The disadvantage is that relating the site coverage to equipment accuracy is not very straightforward. This leads to some restrictive assumptions about the positioning systems used.

F. ASSUMPTIONS MADE IN THE PROGRAM

1. The Coordinate System

A two-dimensional Cartesian coordinate system is used to represent the environment. The units of the coordinates are integer meters. The meter was chosen because it is small

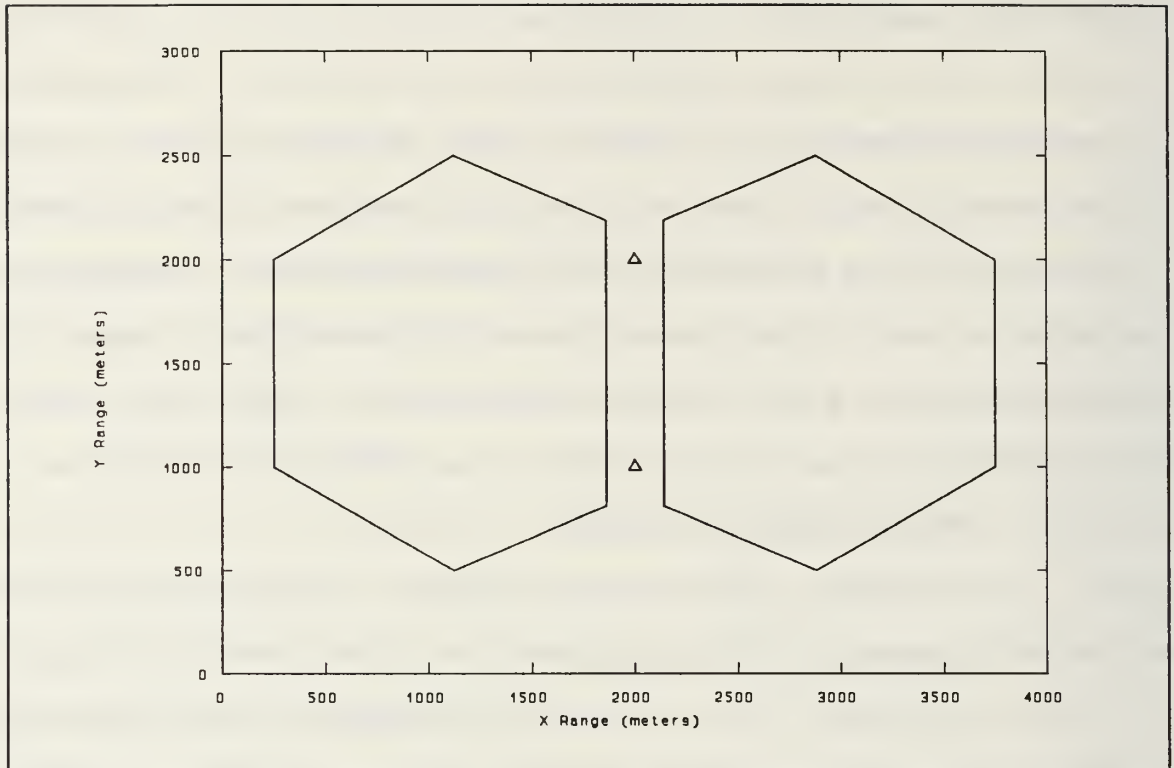


Figure 2. Polygon Approximation of Station Coverage. This is the set of polygons used in the program to approximate the coverage shown in Figure 1. The triangles indicate the positions of the stations.

enough that round-off errors will not make a significant difference in the planning process, and because the program could be easily modified to accept UTM coordinates. The actual coordinates used in the data sets are arbitrary with the origin in the lower left-hand corner.

2. The Search Space

Site locations are only considered at the shoreline. Limiting the search space to essentially one degree of freedom makes the problem much more tractable, and the savings in run time due to a smaller search space and simpler computations

greatly outweigh the added flexibility that would be obtained by extending the search space to the entire land area. In practical experience, almost all navigation sites will be near the shore, although it may occasionally be desirable to place a site on a high hill or peak to increase the effective range of line-of-site equipment.

The search space is further limited by the rectangular chart limits specified in the input data set. The shoreline is clipped at the chart boundary, and the entire survey area should be within these limits. The only consequence of this restriction is that the chart limits specified should be broad enough to include all reasonable site locations.

Even with the above restrictions, the number of possible site locations is infinite. In order to make the problem manageable, we have further restricted the search space with a set of successor heuristics.

3. Positioning System

We assumed only range-range systems would be used in the survey, and made no provisions for a range-azimuth, hyperbolic, or hybrid positioning system. The following model of the range equipment was used:

1. Only one type of station equipment is used for a given survey, and the range limit is constant.
2. The equipment is assumed to be sufficiently accurate for surveying between the 30° and 150° angles of intersection at the scale of the survey.

3. No distinction is made between line-of-sight and ground-wave equipment.
4. Land masses do not affect station coverage.

G. SUMMARY

This chapter begins with a discussion of accuracy requirements for hydrographic surveys, and the practical constraints limiting the survey planner's selection of navigation control sites. The representation of the problem is outlined, as well as the specific assumptions made by the program.

III. DESCRIPTION OF PROGRAM

A. OVERVIEW OF SOFTWARE AND HARDWARE

We developed the program in Arity Prolog. Although the program can be compiled and run as an executable file, all testing was done in the Arity Prolog Interpreter to facilitate the development process. We used an XT compatible Mirage computer with an 8088-10MHz processor and no math coprocessor.

B. SEARCH HEURISTICS DEVELOPED

To aid in the search for the optimum site network, we developed several heuristics, both for evaluating existing states and for generating successor states. These heuristics lead, in most cases, to optimal or near-optimal solution states.

A* search is an improved form of branch-and-bound search. It requires both a cost function that returns the cost of a given state, and an evaluation function which estimates the additional cost required for a solution. An optimal solution is guaranteed if the evaluation function returns a lower bound on the true cost.

The cost of a given state is simply the sum of the costs of each of the sites in the state. To estimate the number of additional sites needed, two fundamental heuristics were developed. The first heuristic looks at the size of the area

not covered by the existing sites, but ignores the actual distribution of this area. The second heuristic looks at the number of isolated areas remaining, but ignores the size of these areas. A combination of these two heuristics proved to be the most useful in directing the search.

We also developed a number of heuristics for defining the successor states. If these heuristics generate too many successors, it would lead to a very long search, but would be more likely to generate the optimal solution to the problem. If, on the other hand, the successor heuristics trim the search space too much, the search time would be shortened but the eventual solution found may not be optimal. Our goal then, is to create a search space which is just large enough to guarantee inclusion of the optimal solution. This proved to be the most critical, and most difficult, problem in the development of our program.

C. MAJOR COMPONENTS OF PROGRAM

1. Data Structures

a. Geometric Structures

A number of geometric structures were defined for the program. The basic unit of all of the geometric structures is the point, defined as a list of two integers $[X,Y]$. Line segments are represented as a list of two points, and are considered to have direction (i.e., $[P1,P2] \neq [P2,P1]$).

Polygons are represented by a list of points with the last point on the list equal to the first. They are defined in a clockwise sense, with the interior of the polygon to the right of the directed list. A fragment is a list of points which does not form a closed polygon (i.e., the last point is not equal to the first).

Single closed polygons are not always sufficient to represent shorelines and survey areas. These objects can be represented either as lists of polygons and fragments, or as lists of segments. Both representations are used in our program.

Lines are represented in the program as a list of three floating-point numbers $[A,B,C]$ such that the equation $Ax+By+C=0$ is satisfied. Rays are represented by a two-element list consisting of a point and a direction in radians. The direction is a floating-point number between 0 and 2π .

b. States

In this search problem, a state would be uniquely defined by a set of navigation sites. We found it convenient, however, to include additional information in the state. States are represented by a list of three elements:

1. The list of navigation sites.
2. The average amount of survey area covered by each site.
3. The portion of the survey area not covered by any site.

The list of sites is represented by a list of points, the average coverage per site is computed as a floating-point number, and the remaining survey area is represented by a list of polygons.

Two lists of sites may be equivalent without being identical. For one thing, the ordering of the list of sites is unimportant. In addition, a list of sites which is very close to an existing list may be considered virtually identical, or equivalent. To test for equivalency of states, a constant parameter called site deviation (SD) is defined. Two lists of N sites to be tested for equivalency are first sorted, then the positions of the respective sites ($P1_i, P2_i$) are compared. The two sets are considered equivalent if the following condition is met:

$$\sqrt{\frac{1}{N} \sum_{i=1}^N |P1_i - P2_i|^2} \leq SD \quad (2)$$

If the site deviation is set too low, the program may waste time examining states which are not significantly different from existing states. If it is set too high, the program may overlook potentially useful states.

2. Input and Output

The only input required of the user is the name of the input data file. This file is a Prolog database which contains facts and rules defining the survey requirements:

1. The chart limits defined by two points: the lower left and upper right corners of the chart. This chart represents the absolute limits of the search space.
2. The shoreline defined as a list of polygons and fragments.
3. The survey area defined as a list of polygons.
4. Existing geodetic control defined as a list of points or the empty list if none exists.
5. The minimum and maximum ranges of the range equipment.
6. A set of rules describing the cost of a site as a function of position.
7. The site deviation for the data set.

The output of the program is a list of recommended navigation sites. The user may obtain alternate solutions by forcing the program to backtrack, but the first solution should be optimal.

3. The Search Algorithm

The search algorithm used is a modification of a problem-independent A* search program developed by Rowe (1988). The major modification to the code involves a test to avoid states already examined. This test now occurs as part of the successor function. Our reason for making this change is that the polygon routines are slow, and we did not want the program to redundantly compute the remaining survey area. The test now occurs after the sites have been selected, and if the set of sites is equivalent to a set previously examined, the successor fails.

We were also able to simplify the test itself--it does not check to see if the same state was found at a lower cost. This is possible since the cost does not depend on the path the program takes to arrive at the state. In fact, the modified program does not keep track of the path-list at all.

The search algorithm as used in the program does not, however, meet the criteria for a true A* search, because the evaluation function is not guaranteed to return a lower bound on the estimated cost of reaching the goal.

Rowe's search program requires that four additional predicates be defined. These are the goal state, the cost function, the evaluation function, and the successor function.

D. THE GOAL STATE

The goal of the search is to find a list of navigation sites that provide adequate coverage for the entire survey area. The search continues until the entire survey area is covered, therefore the goal state is any state in which the remaining survey area is the empty list.

E. THE COST FUNCTION

1. Total Cost

The total cost computed for each state is the sum of the actual network cost ($COST_A$) and an artificial geometric cost ($COST_G$). The primary goal of the program is to minimize $COST_A$, but we chose to add a small geometric cost to

differentiate between otherwise equal solutions. $COST_g$ is designed to reward networks with better geometric configurations, an ideal geometric configuration being one in which all sites are equally spaced.

2. Actual Network Cost

$COST_A$ is the sum of the costs of each of the sites within the network. The cost of each site is the cost of establishing the site plus the cost of maintaining the site for the duration of the survey.

The minimum cost we selected for maintaining a site is one, assuming that there are no security or access problems with the site. There is no upper bound for maintenance cost--the more difficult it is to provide security or resupply the site, the higher the cost.

The minimum cost that we selected for establishing a site is zero, which assumes that this cost negligible compared with the maintenance cost.

For testing purposes, we have assumed that a site cost predicate is provided in the input data set which expresses the cost as a function of geographic position. Although the actual cost is also a function of equipment type and duration of survey, these will be constant for a given project.

3. Artificial Geometric Cost

We define an ideal geometric configuration as one where all sites are separated by the same distance, and the

geometric cost of a network is a measure of how much the network deviates from this ideal. Since the geometric cost should not outweigh the actual network cost, the geometric cost function should have an upper bound of one. We selected a geometric cost function based on the distances between the sites in the network:

$$COST_G = 1 - \frac{\min_{i,j,i \neq j} [D_{i,j}]}{\max_{i,j,i \neq j} [D_{i,j}]} \quad (3)$$

where $D_{i,j}$ is the distance from site i to site j .

It should be noted that two states considered equivalent via equation (2) may have slightly different geometric costs associated with them. Therefore, when evaluating output from different algorithms, we should not take the costs too literally--cost differences of 0.2 or less are probably not significant.

F. THE EVALUATION FUNCTION

The purpose of the evaluation function is to estimate the cost of finding a solution from a given state. The basic form of our evaluation function is

$$EVAL = SE \times NE \quad (4)$$

where SE is the estimated cost of a single site and NE is the estimated number of additional sites needed.

We developed two basic formulas for estimating the number of sites needed. The first is to divide the remaining survey area (RA) by the average coverage per site (AC) in the existing network, assuming each new site will cover that average value. The second is to count the number of disjoint pieces (NP) in the remaining survey area, and assume it will require one site per piece. We then tested two methods of combining the formulas. The first method computes a weighted sum of the two basic formulas:

$$NE = W_1 \cdot \frac{RA}{AC} + W_2 \cdot NP \quad (5)$$

$$W_1 + W_2 = 1$$

while the second method uses the maximum of the two formulas:

$$NE = \max \left(\frac{RA}{AC}, NP \right) \quad (6)$$

The evaluation function based on equation (5) is called EVAL1, and the evaluation function based on Equation (6) is called EVAL2. EVAL1 was tested for several values of the weights W_1 and W_2 , and both functions were tested for several estimates of the site cost.

In general, as the estimated site cost is increased, the number of states examined before a solution is found decreases, resulting in a faster program. Unfortunately, as the estimated site cost is increased, the probability of

arriving at a suboptimal solution also increases. Our goal was to find a value for the estimated site cost which leads to an optimal or near-optimal solution in a reasonably short time.

For comparison purposes, we also wanted an algorithm that would always return a lower bound for the cost estimate. To achieve this, we created another evaluation function called EVAL3. For any non-goal state, the function returns one; for a goal state, zero. This is guaranteed to be a lower bound for the actual cost since the minimum number of sites needed for a non-goal state is one, and the minimum cost per site is one.

Using EVAL3, the program performs a true A* search, and is guaranteed to find the optimal solution. Each of the data sets were processed with EVAL3 to provide a bench mark for comparing the results of the heuristic algorithms.

Implementing EVAL1 and EVAL2 requires computing the area of an arbitrary polygon. Our algorithm divides the polygon into triangles and sums the areas of the triangles. Because the polygons may be concave, some of the triangles may contain area outside the polygon boundary. Since all of our polygons are defined clockwise, we were able to compensate for this by defining a special triangle area function which returns negative areas for counterclockwise triangles. When the polygon is triangulated, exterior triangles will be

counterclockwise, and the sum of the positive and negative areas will be the correct area of the polygon.

G. THE SUCCESSOR FUNCTION

For this problem, a successor could be generated in a number of ways. One or more sites could be added to the existing network, sites could be removed from the network, or existing sites could be repositioned. If all of these ideas were employed, the search space would be too large to be practical. We have therefore limited our successor function to adding sites to the existing network and, in most cases, only a single site is added. We have further limited the search space by using a set of heuristics to select new sites. The essentials of the successor function are shown in Figure 3.

There are essentially two ways to define an ideal pair of sites. One is that the two sites should form an equilateral triangle with the center of the survey area. This would place the center of the survey area at the center of one of the coverage circles. The other is that the two sites should be equidistant from the center of the survey area and form a 90° intersection angle. This would guarantee very good geometry in the neighborhood of the center. Because it is computationally simpler, the latter method is the basis of our heuristics. Chapter IV discusses the successor function heuristics in greater depth.

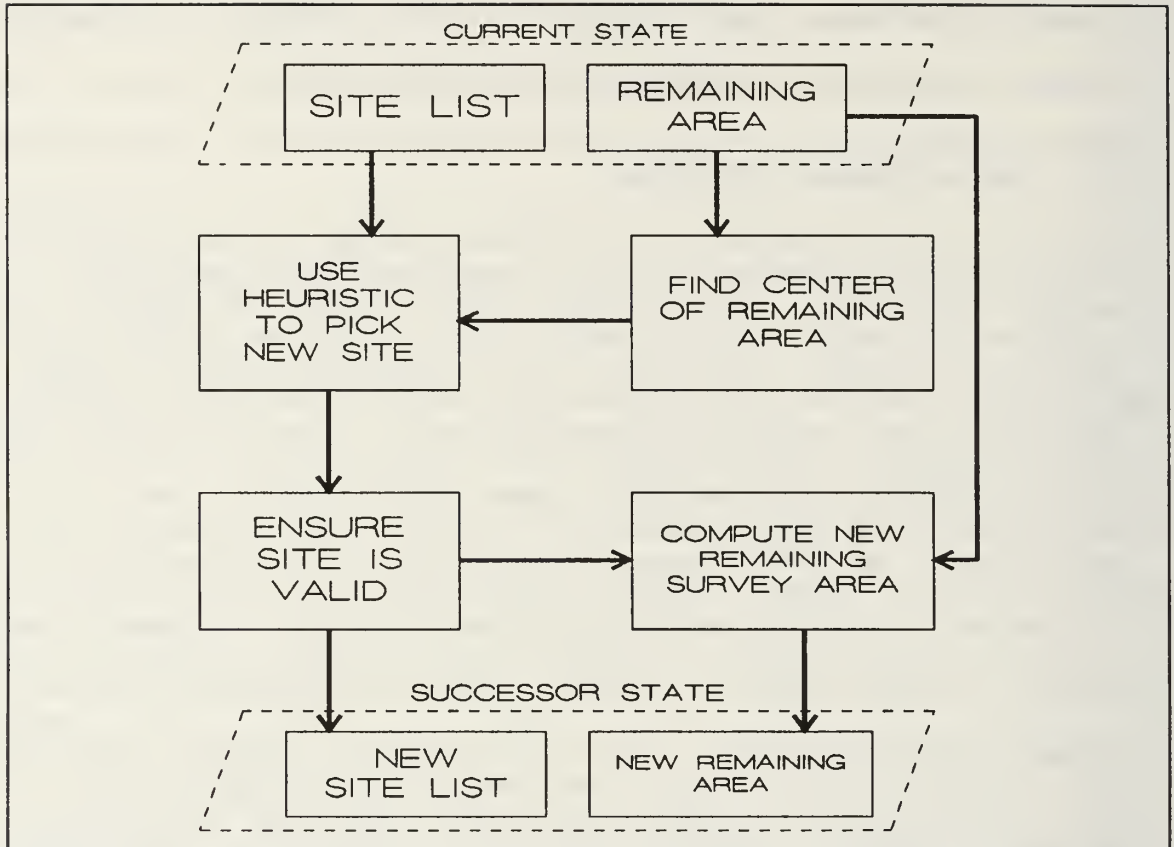


Figure 3. Block Diagram of the Successor Function

H. SUMMARY

Most of the major components of the program are covered in this chapter. The fundamental data structures are introduced, and the overall search strategy briefly discussed. The specific components of the search program are discussed in detail, including the goal state, cost function, and evaluation function. The geometric functions required by these components are also discussed.

Finally, a basic overview of the successor function is given, but an in-depth discussion of this function is left until Chapter IV.

IV. SUCCESSOR FUNCTION HEURISTICS

A. THE BASIC STRATEGY

We developed a set of heuristics to generate the successor states for the search. Each heuristic will generate one or more successors if used, and all are used independently. That is, no heuristic is combined with other heuristics to generate a successor. Our heuristics can be divided into two main classes: initial-site heuristics and subsequent-site heuristics. The initial-site heuristics are

1. Begin at extreme end. Select a site that is close to the vertex of the survey limits farthest from the center of the survey area.
2. Begin in middle. Select a site that is near the center of the survey area.
3. Begin with a geodetic station. Select a site at a known geodetic control station.
4. Begin with two sites. Find two new sites that form nearly a 90° angle of intersection with the center of the survey area.
5. Begin with two geodetic stations. Find two existing geodetic stations that form nearly a 90° angle of intersection with the center of the survey area.

while the subsequent-site heuristics are

6. Add a new site that forms nearly a 90° angle of intersection with an existing site and the center of the remaining survey area.
7. Add a site down the coast from an existing site.

8. Add a site at a geodetic station. Find the existing geodetic station that forms an angle of intersection nearest to 90° with the last site picked and the center of the remaining survey area.
9. Add a site at half the maximum range. If the dimensions of the survey area are large compared with the range of the equipment used, choose a site that is half the maximum range limit from the last site picked.

The initial-site heuristics generate successors for a state which has no navigation sites. Heuristics 1, 2, and 4 will each generate one successor. Heuristic 3 will generate as many successors as there are geodetic stations, and heuristic 5 will generate one successor if there are at least two geodetic stations.

The subsequent-site heuristics generate successors for states with one or more sites. Heuristics 6 and 7 will each generate up to two successors for a state with one site, and up to one successor for each existing site for a state with more than one site. Heuristic 8 will generate one successor if there are any geodetic stations which have not already been used for sites. Heuristic 9 could generate any number of successors if the dimensions of the survey area are greater than half the range of the positioning equipment.

B. CHOOSING THE INITIAL SITE

In the early phases of program testing, we tried several random initial sites for each of the data sets. Our purpose was to determine how critical initial site selection is in

finding an optimal solution, and to look for patterns in the test results that would help us develop a heuristic for initial site selection. The results of this testing indicate that initial site selection is fairly critical, but we were unable to find a heuristic that would produce optimal solutions in all cases. We decided to use several heuristics to provide a good field of initial states.

C. CHOOSING SUBSEQUENT SITES

The successor function should find the point on the shoreline that best approximates the ideal site--one which is the same distance from the center of the survey area as an existing site and forms a 90° intersection angle. Unfortunately, an exact solution of this requires solving a set of nonlinear equations. Even if the equations were linearized by a Taylor series, the computations would be impractical for our program. What we have done instead is compute an ideal site location, then find the point on the shoreline nearest that location.

Originally, we computed two ideal site locations for each site in the current state. We developed the following equations for computing the coordinates of the ideal site location (X_I, Y_I) from the coordinates of the existing site (X_E, Y_E) and the center of the survey area (X_C, Y_C) :

$$\begin{aligned} X_I &= X_C \pm (Y_C - Y_E) \\ Y_I &= Y_C \mp (X_C - X_E) \end{aligned} \quad (7)$$

These points will be the same distance from the center of the survey area as the existing site, and form a 90° intersection angle. Preliminary tests revealed that the points computed from equation (7) do not always lead to good shore sites, especially for open coast surveys. The nearest point on shore may not be a good site. We then added two more site location definitions. The equations for the new points are:

$$\begin{aligned} X_I &= X_E \pm 1.5(Y_E - Y_C) \\ Y_I &= Y_E \mp 1.5(X_E - X_C) \end{aligned} \quad (8)$$

We designed equation (8) to obtain reasonably good shore sites for open coast surveys, rather than sites with a 90° intersection angle. Figure 4 shows the positions of the ideal and actual sites for one of our test cases.

If the survey area is large compared to the range of the radio positioning equipment, the actual coverage of two sites may be less than their geometric coverage. This occurs if two sites are separated by a distance greater than half the maximum range of the equipment. Figure 5 illustrates a geometric coverage clipped by the range circles of the sites. If this is the case, the heuristics described above may not produce good shore sites. To solve this problem, we introduced another heuristic, which finds sites at a distance

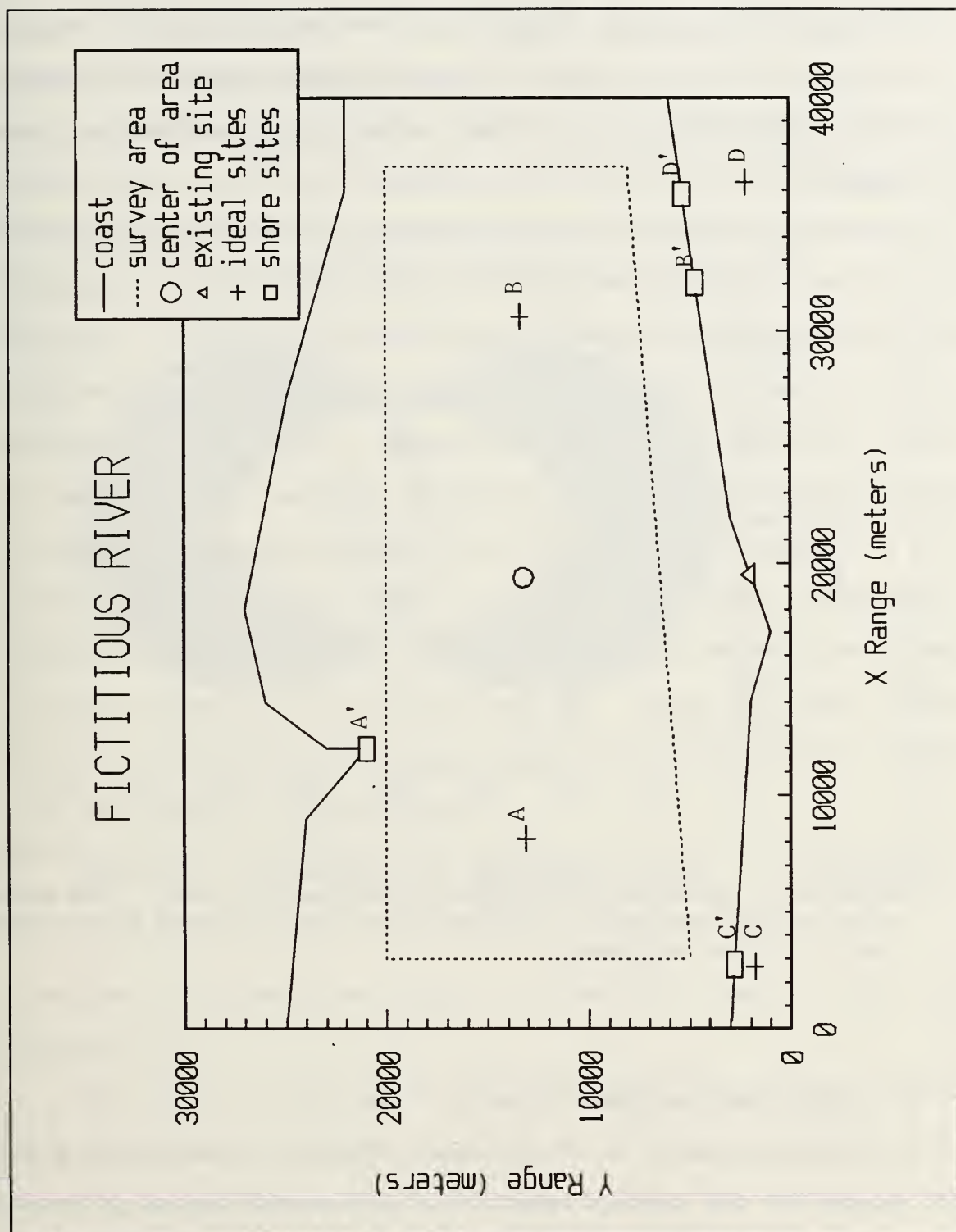


Figure 4. An Example of Site Selection Heuristics for the RIVER Data Set. Sites A and B were computed with equation (7), and sites C and D were computed with equation (8). A', B', C', and D' are the resulting shore sites.

of half the maximum range from an existing site. This is essentially the optimum distance between sites--it produces the largest coverage area that is not clipped by the equipment range.

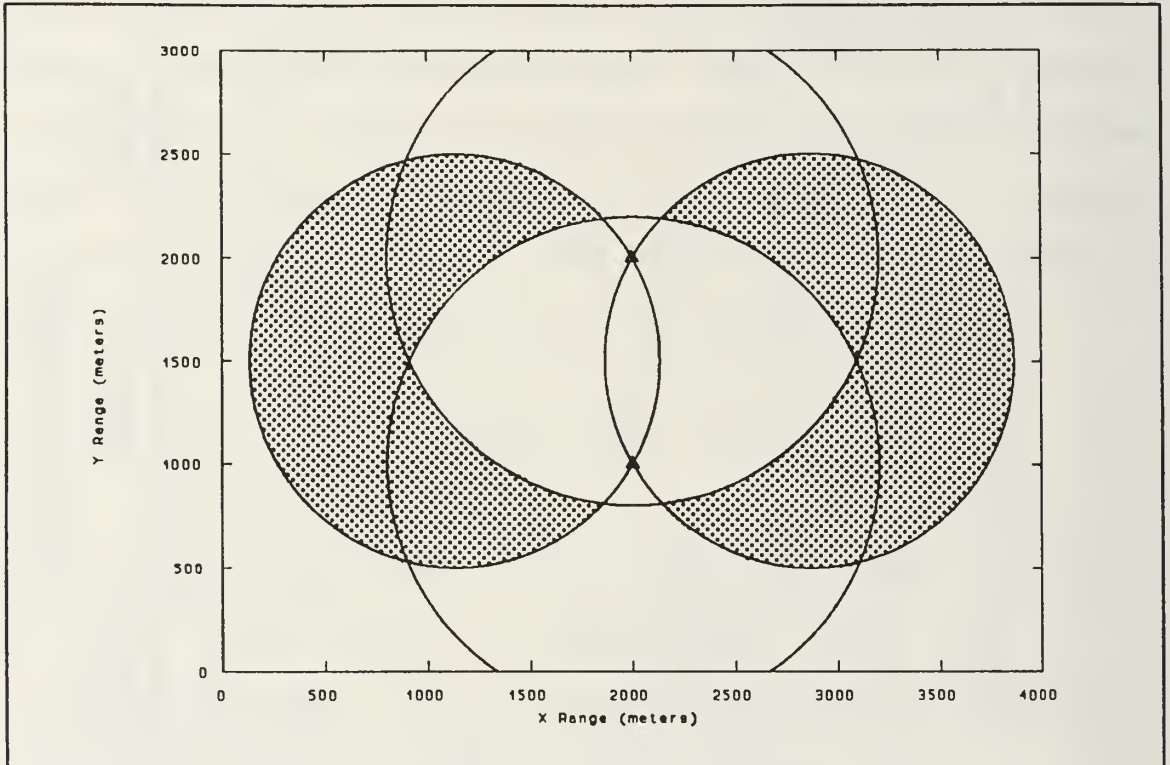


Figure 5. Range Clipping of Geometric Coverage. The triangles indicate the positions of the stations. The shaded areas of the geometric coverage have been clipped by the range limits of the equipment.

D. APPLYING THE HEURISTICS

Heuristics 4, 5, 6, 7, and 8 require an estimate for the center of the survey area. The major problems with this are that the remaining survey area may be represented by multiple polygons, and those polygons may not be convex. We decided to

use a function that would find the center of the largest polygon in the survey area.

We originally decided to use a combination of two functions. The first computes the center of the polygon by taking the mean of all the vertices. If the polygon is not convex, however, the point returned may not be inside the polygon. In this case, the second function triangulates the polygon and returns the center of the largest triangle in the polygon. We later developed another algorithm which triangulates the polygon, computes the area and center of each triangle, and takes the mean of these points, weighted by the triangle areas. Heuristics 4, 5, 6, 7, and 8 also require a measure of how good the intersection angle between two sites and the center of the survey area is. Since the ideal angle is 90° , we chose the following equation for the quality (Q) of the intersection angle (β):

$$Q(\beta) = (\beta - 90^\circ)^2 \quad (9)$$

The best intersection angle is then the angle with the smallest Q.

Heuristic 4 requires finding two sites with a good angle of intersection. A list of prospective sites is generated by drawing radial lines at 10° increments from the center of the survey area and finding the points where these rays intersect

the shoreline. This list of points is then searched to find which two form the best intersection angle.

Heuristic 6 is applied via equation (7) and heuristic 7 is applied via equation (8). If only one site is in the current state, these heuristics will each generate two successors. If two or more sites are in the current state then one successor is generated for each site. Either heuristic 6 or heuristic 7 is used, depending on which heuristic generates a better site based on equation (9).

E. UPDATING THE REMAINING SURVEY AREA

Once the new site has been selected, the program must determine the remaining survey area. To accomplish this, three polygon routines were developed: intersection, union, and clip_intersection. The first two are self-explanatory. The third subtracts the intersection of two polygons from the first polygon, and is represented symbolically as

$$P_{CLIP} = P_{WHOLE} - (P_{WHOLE} \cap P_{TEST}) \quad (10)$$

where P_{CLIP} is the result of clipping P_{WHOLE} by P_{TEST} .

A number of polygon routines for convex polygons have been previously developed. Plastock and Kalley (1986) describe a method of clipping an arbitrary polygon with a convex polygon. In our application, however, we are not guaranteed that any of

our polygons are convex, so we were forced to develop something different.

All three of our routines use similar ideas. Polygon lists are converted into segment lists, then the intersections of the segments are computed and a new segment list is built. This can then be transformed into a polygon list. The major weakness of the routines is the inability to handle the case where two line segments intersect in another line segment--the intersection must be a point.

To find the survey area remaining after a new site is added, the program generates the coverage added by the new site and clips this from the existing survey area. The coverage (C_{TOTAL}) added by the new site (N) is given by

$$C_{TOTAL} = C_{1,N} \cup C_{2,N} \cup \dots \cup C_{N-1,N} \quad (11)$$

where $C_{i,j}$ is the area covered by sites i and j . For the simple case where $C_{i,j}$ is equal to the geometric coverage ($GC_{i,j}$), the coverage polygons are obtained by transforming a prototype coverage polygon. This transformation is similar to those described by Rogers and Adams (1990) and involves scaling, rotating, reflecting, and translating the polygon point set.

The remaining survey area (RA) is then

$$RA = EA - (EA \cap C_{TOTAL}) \quad (12)$$

where EA is the existing survey area. We originally used equations (11) and (12) to find the remaining area, but encountered some difficulty. If $GC_{i,j}$ is clipped by the range limits of i and j , the coverage is given by

$$C_{A,B} = GC_{A,B} \cap RL_A \cap RL_B \quad (13)$$

where RL_i is the range limit of site i . The range limit polygons are obtained by performing a point transformation on a range prototype polygon. Equation (11) then becomes

$$C_{TOTAL} = (GC_{1,N} \cap RL_1 \cap RL_N) \cup \dots \cup (GC_{N-1,N} \cap RL_{N-1} \cap RL_N) \quad (14)$$

The redundancy of RL_N in equation (14) results in colinear line segments which cause a failure in the polygon union function. One way to solve the problem is to rewrite equation (11) as

$$C_{TOTAL} = (GC_{1,N} \cup \dots \cup GC_{N-1,N}) \cap (RL_1 \cup RL_2 \cup \dots \cup RL_N) \quad (15)$$

thus avoiding the redundancy of RL_N . Another way to solve the problem is to rewrite equation (12) as

$$RA = EA - (EA \cap C_{1,N}) - (EA \cap C_{2,N}) - \dots - (EA \cap C_{N-1,N}) \quad (16)$$

which avoids the polygon union altogether. Equation (16) is used in the final version of our program.

F. THE SEARCH SPACE

Figure 6 shows the first three levels of the search space for a simple data set with no range constraints and no existing geodetic control (i.e., heuristics 3, 5, 8, and 9 are not used). For this case, the only subsequent-site heuristics used are heuristics 6 and 7, which generate four successors for a state with one site, and as many successors as there are sites for states with multiple sites. The following equation expresses the number of states (NS) at the L th level of a search tree which begins at level 0 with one state containing S sites:

$$NS(L, S) = \frac{(L + S - 1)!}{(S - 1)!} \quad (17)$$

For the case shown in Figure 6, level 1 contains two states, level 2 contains nine states, and the total number of states (TNS) for each subsequent level (L) is given by:

$$\begin{aligned} TNS(L) &= 9NS(L-2, 2) \\ &= 9(L - 1)! \end{aligned} \quad (18)$$

It is clear why EVAL3 is not feasible, especially for surveys requiring many sites. For example, a six-site solution would require a search of between 200 and 1000 states.

G. SUMMARY

This chapter covers the basic heuristics which govern the selection of successor states. Two general classes of heuristics are presented: Initial site heuristics and subsequent site heuristics.

The polygon functions used to generate the site coverage and trimming of the survey area are described. A problem with these functions is identified--namely, the inability to handle colinear line segments--and a procedure for rewriting the algorithms to avoid this problem is outlined. Finally, the factorial expansion of the search space is explored.

V. RESULTS OF TESTING

A. THE TEST DATA SETS

There are many different types of surveys that we attempted to accommodate in the program, each with its own specific problems and requirements. We selected six primary and five secondary data sets to test the program.

The six primary data sets are fictitious areas constructed to test the program performance in idealized scenarios. They are COAST, BAY, RIVER, ISLES, POINT, and MIXED, and represent an open coastline, a bay, a river, a group of islands, a point, and a mixture of coastline and islands. These six data sets were each tested under three different cases:

1. The range of the equipment is very long with respect to the dimensions of the survey and there is no existing geodetic control in the area.
2. The range of the equipment is long and there are three existing geodetic control stations in the area.
3. The range of the equipment is very restricted and there is no existing geodetic control in the area.

The primary data sets, along with solutions, are displayed graphically in Appendix A.

The five secondary data sets were taken from actual charts, and each was tested under its own specific requirements. MONBAY is a coastal survey of the entire

Monterey Bay. In this survey, medium-range positioning equipment (ARGO) is used, and no existing geodetic control is assumed. MONHAR is a survey of Monterey Harbor conducted with short-range equipment (Miniranger) with several existing geodetic control stations. OBISPO is a harbor and coastline survey of San Luis Obispo Bay with short-range equipment (Miniranger) and no existing control. SUISUN is a channel survey in Suisun Bay with short-range equipment (Miniranger) and no control. Finally, HELENA is a survey around the Island of Saint Helena with short-range equipment (Trisponder) and no existing control. The secondary data sets and solutions are displayed graphically in Appendix B.

In all of the secondary data sets, the maximum equipment ranges were obtained from the NOAA Hydrographic Manual (Umbach, 1976). In the primary data sets, fictitious ranges were used.

B. THE SUCCESSOR HEURISTICS

1. The Center of Polygon Function

Section IV.D. contains a description of the two methods we developed for finding the center of an arbitrary polygon. The first method uses both the convex and concave functions, while the second method employs a weighted mean. Figure 7 shows the centers obtained for a set of polygons using the convex, concave, and weighted functions.

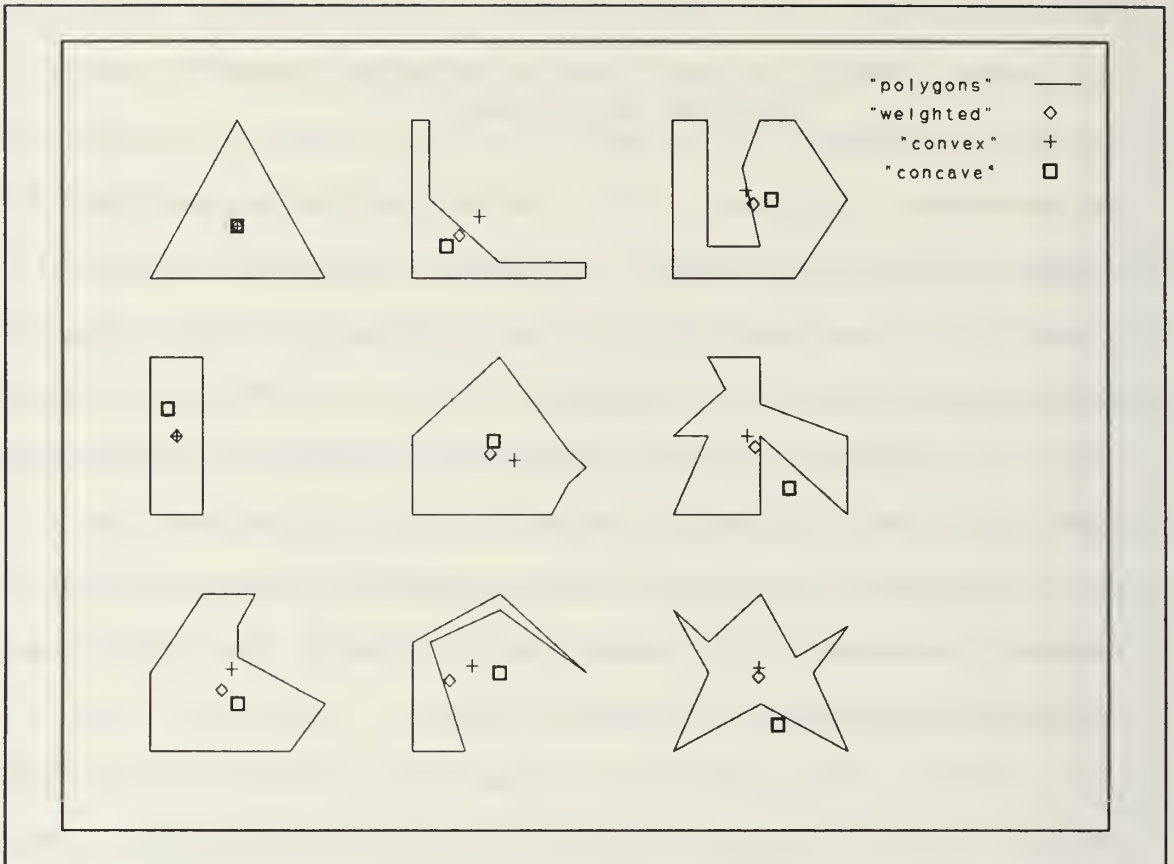


Figure 7. Three Center of Polygon Functions. The centers of nine test polygons are shown, as determined by the weighted, convex, and concave functions.

We were surprised to discover that the concave function does not perform as we intended. We developed it to ensure that the center was within the polygon boundary, but the method we used to triangulate the polygon can result in a center outside the polygon.

The weighted sum appears to be the best method, but we decided to run EVAL3 for the primary data sets with both methods to determine which one leads to a lower solution cost.

2. The Optimal Solution

As mentioned previously, EVAL3 is an A* algorithm and guaranteed to return the optimal solution in a search space. Unfortunately, this may not correspond to the optimal solution in the real world. Because the search space is limited by the successor heuristics--and because relative costs are approximate--the "real" optimal solution may not be included in the search space. In our subsequent discussion, the term optimal solution will refer to the best solution in the search space, while the term true optimal solution will refer to the best solution in the real-world situation. The term near-optimal refers to those solutions which differ from the optimal by 0.2 "cost units" or less, and the term suboptimal will refer to solutions which differ from the optimal by greater than 0.2 in cost.

It is very difficult to determine if our search space includes the true optimal solution. We decided to test the program by comparing the output of EVAL3 to a manual solution obtained by a human survey planner. The manual solution was obtained by drawing sites on graphic printouts of the data sets, and testing the solutions by drawing range and coverage circles with a compass. Additionally, the human was allowed to take as much time as needed, and, in some cases, had seen the computer's solutions (the computer tests and manual tests were performed concurrently). While this is not entirely satisfactory--the human is still not guaranteed to find an

optimal solution--it does give us some idea of the performance of the successor heuristics. Table I lists the results of these tests. Note that version 1 of EVAL3 uses our convex/concave center of polygon algorithm, while version 2 uses the weighted mean algorithm.

The results show that, for most of the data sets, our program does indeed return a solution similar or better in cost to that of a manual search. The most notable exception is the COAST data set. This may very well indicate that our successor heuristics are inadequate for the open coast situation.

Table II shows a similar set of solution costs for the secondary data sets. Only version 2 of EVAL3 was used in this case, because the weighted mean is clearly a better center of polygon function. Notice that both the computer search and manual search failed on the HELENA data set. We inadvertently introduced a set of survey requirements which cannot be satisfied. Otherwise, the results of the secondary data set tests are positive.

3. Heuristics Used in Optimal Solutions

Table III lists the various heuristics we developed and the number of times each heuristic was used in the optimal solutions for the primary data sets.

Table I. EVAL3 TEST FOR PRIMARY DATA SETS

DATA SET	SOLUTION COST		
	EVAL3 version 1	EVAL3 version 2	Manual Search
COAST case 1	5.03	6.73	3.26
COAST case 2	5.28	4.21	2.50
COAST case 3	6.70	6.68	3.26
BAY case 1	4.70	6.70	4.80
BAY case 2	3.83	3.90	3.90
BAY case 3	6.53	5.03	6.62
RIVER case 1	4.59	4.61	4.79
RIVER case 2	3.26	3.26	3.26
RIVER case 3	6.56	6.53	4.80
ISLES case 1	4.67	4.63	4.97
ISLES case 2	3.30	3.30	3.30
ISLES case 3	6.53	6.53	4.78
POINT case 1	4.70	4.63	4.51
POINT case 2	3.88	3.65	3.70
POINT case 3	6.57	4.55	6.77
MIXED case 1	6.60	5.20	6.27
MIXED case 2	5.25	5.26	5.25
MIXED case 3	6.39	6.36	6.27
AVERAGE	5.25	5.14	4.68
AVERAGE excluding COAST	5.16	4.99	5.05

Table II. EVAL3 TEST FOR SECONDARY DATA SETS

DATA SET	SOLUTION COST	
	EVAL3 version 2	Manual Search
MONBAY	4.80	4.89
MONHAR	2.50	4.04
OBISPO	4.80	6.71
SUISUN	4.71	4.83
HELENA	failed	failed

The data indicate that the heuristics which begin the search with two sites do not often lead to optimal solutions. Note that the geodetic control heuristics are only used in case 2, while the half range heuristic is only used in case 3. The data seem to support the hypothesis that heuristics 1, 2, 3, 6, 7, 8 and 9 are needed; while heuristics 4 and 5 are not.

C. THE EVALUATION FUNCTION

We tested both EVAL1 and EVAL2 using case 1 of the primary data sets. As Figure 8 and Figure 9 indicate, EVAL2 led to quicker solutions for the lower values of SE. This was not only true on the average, but for every test case run. Although the minimum search time for EVAL1 varied somewhat as a function of W1 and W2, the search time for EVAL2 for SE=1.0 and SE=1.5 was always lower. EVAL1 was sometimes faster with SE=3.0 and SE=10.0, but in these cases both EVAL1 and EVAL2 often led to suboptimal solutions.

Table III. HEURISTICS USED IN OPTIMAL SOLUTIONS

SUCCESSOR HEURISTICS	NUMBER OF TIMES HEURISTIC WAS USED IN OPTIMAL SOLUTIONS		
	Case 1	Case 2	Case 3
Begin at extreme end	0	0	1
Begin in middle	2	1	5
Begin with a geodetic station	0	3	0
Begin with two sites	0	0	1
Begin with two geodetic stations	0	0	0
Add a site at a 90° angle	11	3	7
Add a site down the coast	0	0	0
Add a site at a geodetic station	0	0	0
Add a site at half the maximum range	0	0	6

EVAL2 also has the advantage that the user does not need to select values for the weights W_1 and W_2 . Because EVAL2 performed better than EVAL1, and because of the length of time needed to test all the cases for EVAL1, we elected not to test EVAL1 for cases 2 and 3.

Figure 10 and Figure 11 indicate that the behavior of EVAL2 is very similar in cases 2 and 3. Although increasing SE produces a faster search, an optimal solution is only guaranteed if SE is selected as the minimum cost for an individual site, exactly as expected.

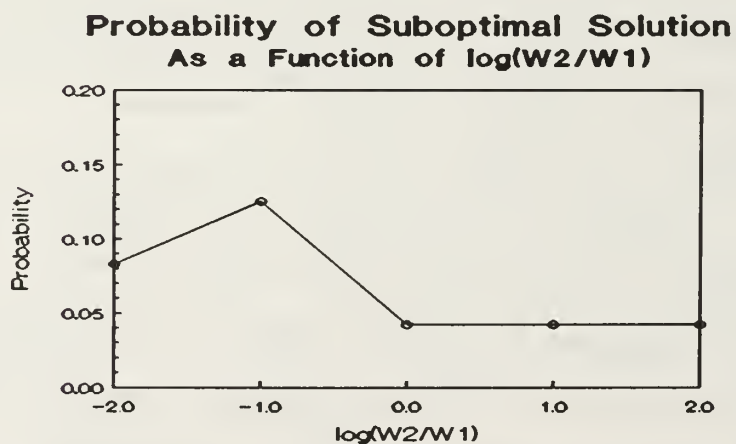
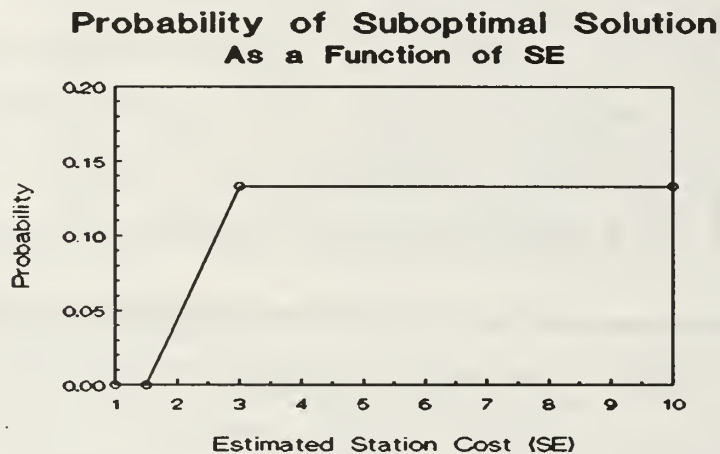
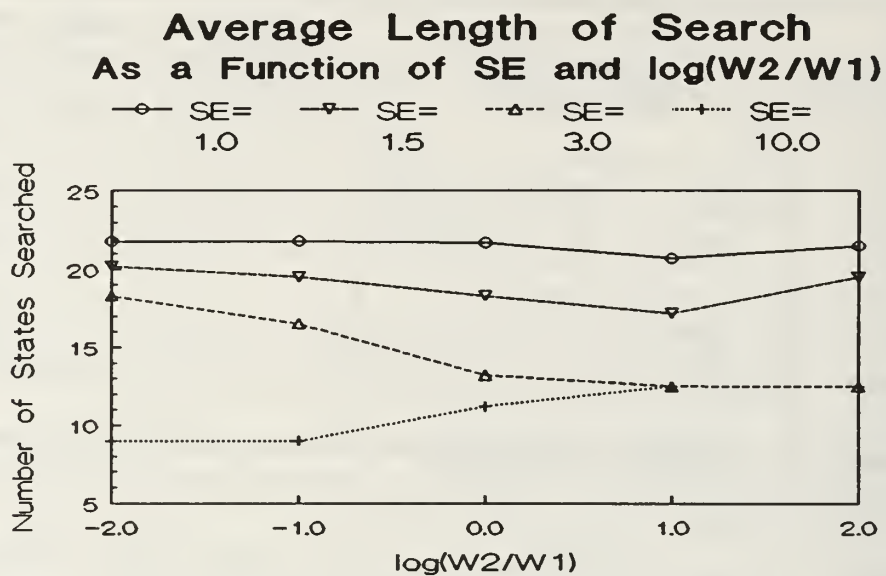
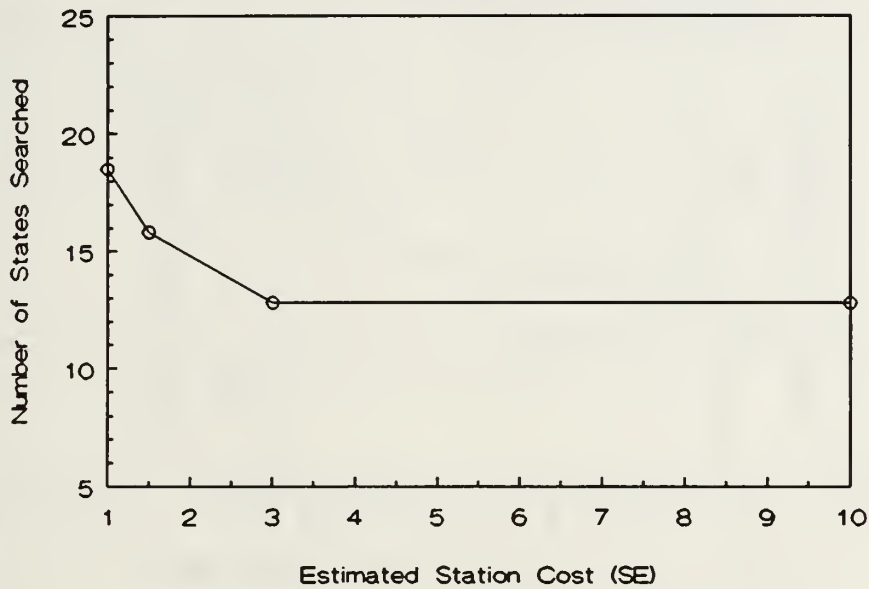


Figure 8. EVAL1 Data for Primary Data Sets Case 1

Average Length of Search As a Function of SE



Probability of Suboptimal Solution As a Function of SE

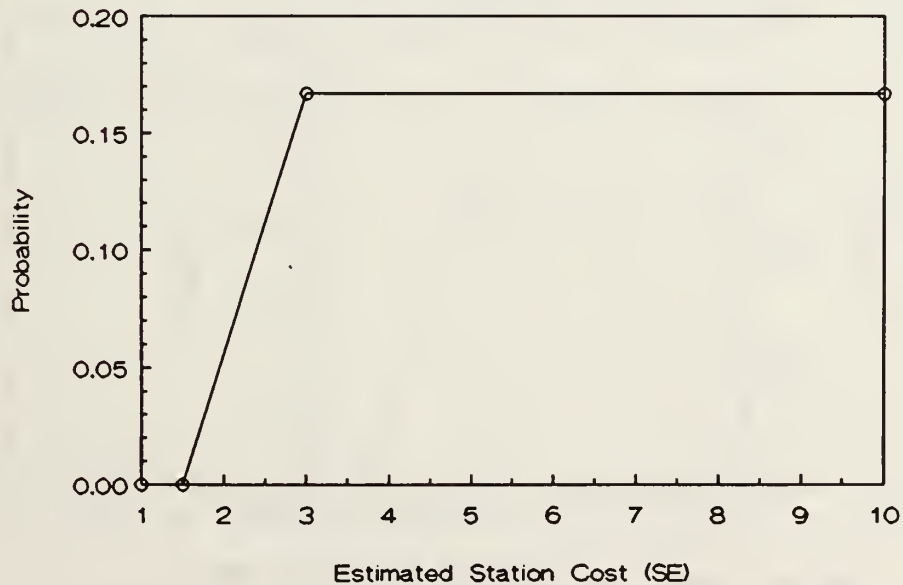
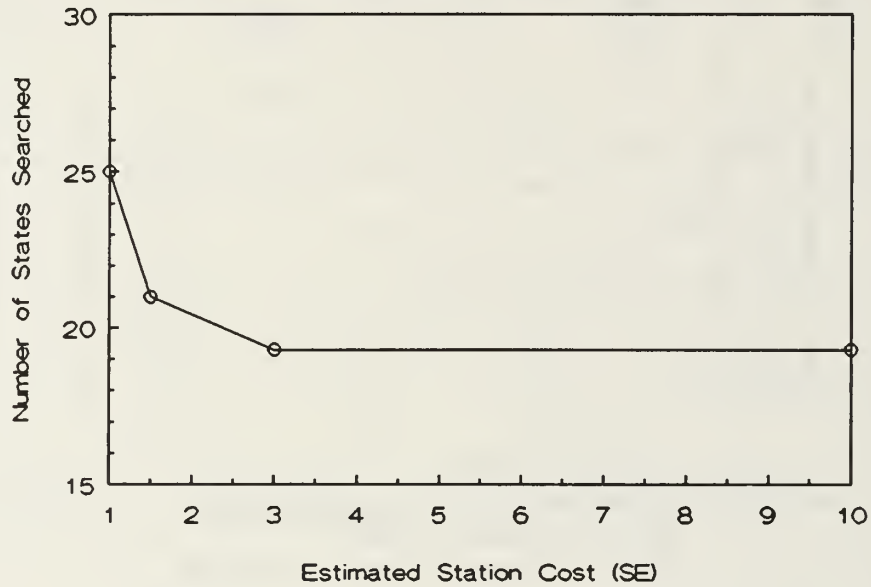


Figure 9. EVAL2 Data for Primary Data Sets Case 1

Average Length of Search As a Function of SE



Probability of Suboptimal Solution As a Function of SE

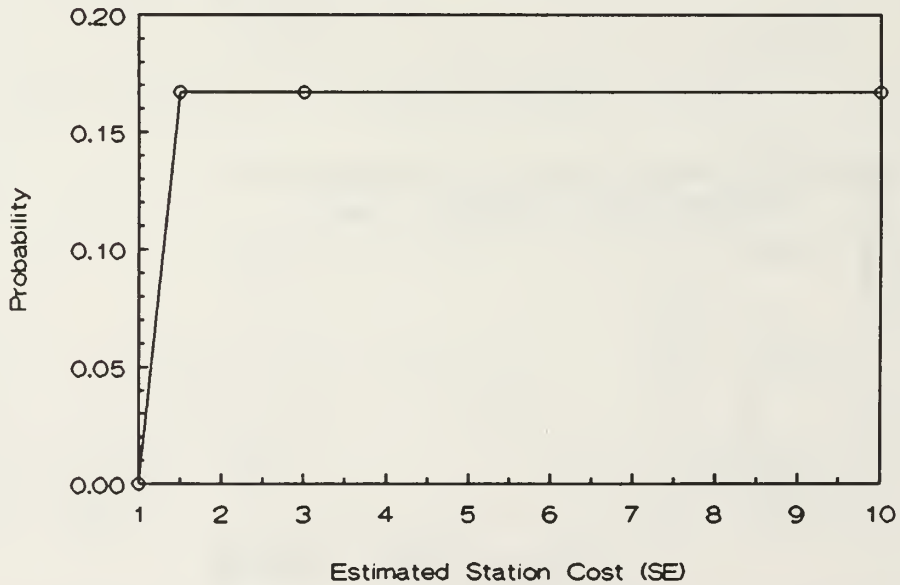
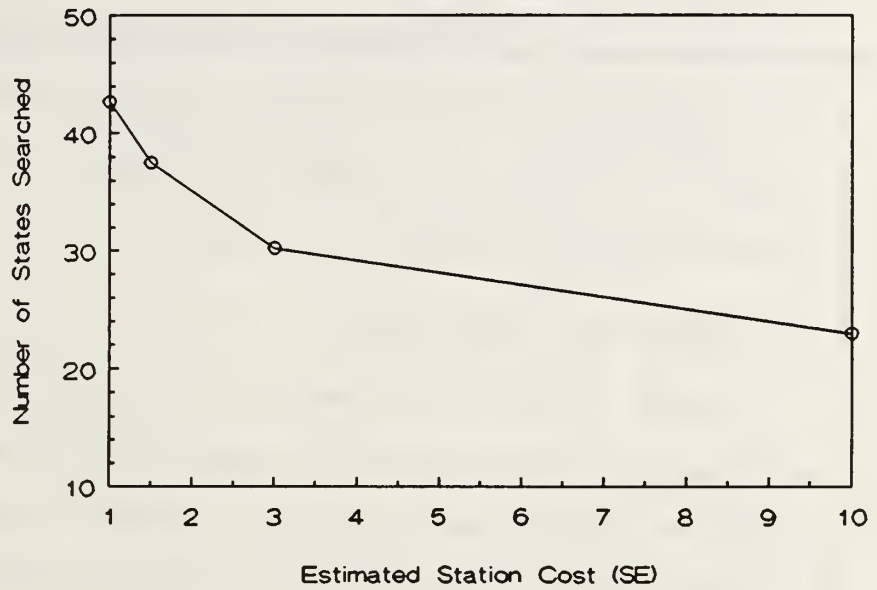


Figure 10. EVAL2 Data for Primary Data Sets Case 2

Average Length of Search As a Function of SE



Probability of Suboptimal Solution As a Function of SE

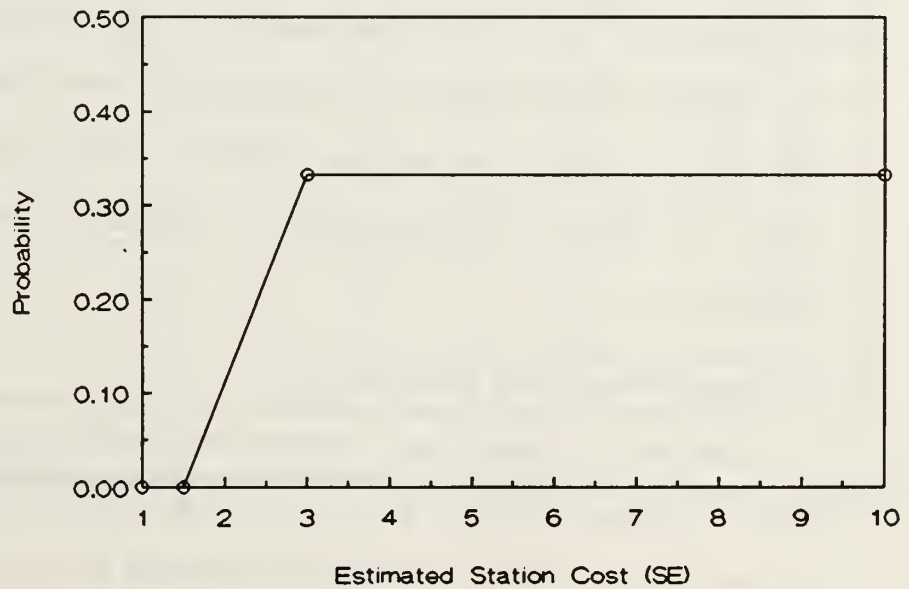


Figure 11. EVAL2 Data for Primary Data Sets Case 3

Table IV compares the number of states searched and the time required for the search for EVAL1 and EVAL2 to that required by EVAL3 for each of the data sets. These results demonstrate how effective the heuristic evaluation functions are in pruning the search space. We set $W1=0.1$, and $W2=0.9$ in EVAL1 and $SE=1.0$ in both EVAL1 and EVAL2, because these values always produced optimal solutions.

D. SPACE AND TIME USAGE

Table V shows the amount of memory space used by the program for the various data sets, and the amount of real time in minutes required to reach a solution. The memory usage for loading just the Prolog interpreter is 40K, and the memory used by the interpreter and our program without running any data sets is 88K.

The memory space used in running the data sets was fairly uniform, and in all of the cases it was very small compared to the memory used by the interpreter and the program itself. The stack usage also varied little. All of the data sets used close to 64K of global stack and approximately 3K of local stack.

E. THE SECONDARY DATA SETS

Table VI lists the results of testing the secondary data sets. These data sets were tested with both EVAL1 and EVAL2. We set $SE=1.0$ for these tests because this should guarantee an

Table IV. EFFECT OF HEURISTICS ON SEARCH SPACE AND TIME

DATA SET	EVAL1:EVAL3		EVAL2:EVAL3	
	STATES	TIME	STATES	TIME
COAST case 1	0.66	0.44	0.48	0.38
COAST case 2	0.66	0.41	0.66	0.38
COAST case 3	0.35	0.26	0.33	0.24
BAY case 1	0.65	0.51	0.57	0.38
BAY case 2	0.66	0.39	0.50	0.30
BAY case 3	0.36	0.26	0.38	0.24
RIVER case 1	0.66	0.45	0.66	0.45
RIVER case 2	0.92	1.00	0.50	1.00
RIVER case 3	0.36	0.26	0.38	0.22
ISLES case 1	0.77	0.66	0.77	0.89
ISLES case 2	0.91	0.91	0.91	0.89
ISLES case 3	0.78	0.61	0.70	0.50
POINT case 1	0.92	0.26	0.62	0.29
POINT case 2	0.50	0.41	0.59	0.41
POINT case 3	0.41	0.26	0.38	0.24
MIXED case 1	0.50	0.31	0.34	0.14
MIXED case 2	0.85	0.76	0.75	0.55
MIXED case 3	0.41	0.30	0.33	0.19
AVERAGE	0.61	0.47	0.54	0.42

Table V. MEMORY USAGE AND RUN TIMES FOR PRIMARY DATA SETS

DATA SET	MEMORY (Kbytes)			TIME (Minutes)		
	EVAL1	EVAL2	EVAL3	EVAL1	EVAL2	EVAL3
COAST case 1	96	96	108	84	71	189
COAST case 2	92	96	100	30	36	82
COAST case 3	100	104	124	213	104	830
BAY case 1	96	96	108	23	17	45
BAY case 2	92	96	100	30	23	77
BAY case 3	100	104	124	255	243	1019
RIVER case 1	96	96	108	18	17	38
RIVER case 2	92	96	100	18	11	18
RIVER case 3	100	104	128	217	174	785
ISLES case 1	96	96	108	55	55	81
ISLES case 2	92	96	100	32	30	35
ISLES case 3	100	104	128	532	438	879
POINT case 1	96	96	108	11	11	74
POINT case 2	92	96	100	30	30	74
POINT case 3	100	104	128	84	59	247
MIXED case 1	96	96	108	30	11	126
MIXED case 2	96	96	100	70	51	92
MIXED case 3	104	104	128	387	247	1295

optimal solution. For EVAL2, we set $W1=0.1$ and $W2=0.9$ because this should give us the minimum search time for $SE=1.0$, based on the earlier tests. EVAL3 was also included in the table.

Table VI. SECONDARY DATA SET TEST RESULTS

DATA SET	NUMBER OF STATES SEARCHED		
	EVAL1	EVAL2	EVAL3
MONBAY	17	14	23
MONHAR	19	15	15
OBISPO	27	20	60
SUISUN	19	17	29
HELENA	failed	failed	failed

In all of the test cases except HELENA, both functions returned optimal solutions. These results support our earlier conclusions based on the primary data sets: Using EVAL2 results in a shorter search than EVAL1, and setting SE to the minimum site cost results in an optimal solution.

F. SUMMARY

We tested two methods of finding the center of a polygon and the weighted mean algorithm performed better. EVAL3 was tested on all of the data sets. We compared the results of these tests to the solutions obtained manually and discovered that the program's solutions compared favorably to the manual solutions in all but the COAST data set.

We then examined the actual successor heuristics used in the search, and discovered that the heuristics which begin the search with two sites are not very useful.

EVAL1 and EVAL2 were then tested on the primary data sets. EVAL2 resulted in a faster search. Both EVAL1 and EVAL2 returned suboptimal solutions for higher values of SE.

The pruning of the search space by the heuristic evaluation functions was discussed. On the average, they reduce the search space by half while returning an optimal solution. The memory usage and running time of the program were also examined.

Finally, we tested EVAL1 and EVAL2 on the secondary data sets. These test results were very similar to the results of the primary data set tests. The HELENA data set proved to have no solution, either with the program or by manual search.

VI. CONCLUSIONS

A. GENERAL CONCLUSIONS

Our tests demonstrate the feasibility of using heuristic search to select sites. With the exception of the COAST data set, the program's solutions are, on the average, at least as good as the manual search by a human survey planner. The memory space used by the program is relatively small and the search times are fairly reasonable, considering the type of computer used in testing. The run times would be greatly reduced by compiling the program and using a better computer (such as a 386 or 486) with a math coprocessor. In situations where it is possible to locate sites in a nearly equilateral triangle, the program performs very well.

The problems with the COAST data set make it clear, however, that additional work needs to be done on the successor heuristics. It is difficult to find a good method of selecting sites for an open coast because it is so different from an ideal survey situation. Heuristic 7, which we designed for this case, is not adequate by itself. Both versions of EVAL3 found much poorer solutions than the manual search. What is perhaps even more surprising is that, for case 1 of the COAST data set, version 1 of EVAL3 found a better solution than version 2. Since it is clear that

version 2 employs a better algorithm for finding the center of a polygon, this may mean that focusing the search on the center of the remaining survey area is not a good strategy--at least for the open coast situation. A possible thesis topic for future students could be the improvement and refinement of the successor heuristics.

B. ADDITIONAL RELATED WORK

There are many other aspects of survey planning which may lend themselves to programs of this sort. These include selecting sites for tide gages; dividing the survey area into the minimum number of standard sheets; and laying out track lines for the survey vessel. Finding the optimal path for a survey vessel to cover a given set of track lines would be very similar to the classic traveling salesman problem, and could possibly be solved through a heuristic search or a simulated annealing technique. These would all make excellent topics for future theses.

C. MODIFICATIONS TO THE HEURISTIC SEARCH PROGRAM

1. Shifting Existing Sites

It may be possible to obtain better results from the program if a successor is generated by shifting one or more of the sites in an existing state. The problem with this is that it is seldom clear which sites to shift in what direction to improve the coverage. If this type of heuristic were used too

liberally, the search space would become too large to be practical. It may be possible to add a post-processing phase to the program, where the best four or five solutions are tested to see if they can be improved in this manner.

2. Fixed Search Space

Instead of using the successor heuristics to limit the search space, the space can be limited by beginning with a fixed set of possible station sites, each with a fixed cost associated with it. A successor would be obtained by adding any site on the list which is not already in the current state. The only heuristics employed in this program would be the evaluation heuristics. This type of program assumes that a survey team has already performed an initial reconnaissance of the area, selected a number of possible station sites, and estimated the relative cost of each site.

3. Grid Approximation

As mentioned previously, an alternative to approximating the coverage and survey area with polygons is to approximate the survey area with a set of grid points. This method has the advantage of selecting sites based on the actual horizontal accuracy at each grid point, rather than on the 30° to 150° coverage circles. Because of this, it would lend itself easily to other control methods such as range-azimuth and hyperbolic, and would allow different sites to have different equipment (with different accuracies).

The major problem with this method is that it could require considerably more space and time to run since the grid must be sufficiently fine to find any coverage gaps. Another problem is that it would be difficult to apply most of our successor heuristics (which rely on the center of polygon function), and our most important estimation heuristic (which counts the number of polygons in the remaining survey area). It is possible that different heuristic could be developed for this case. The successor heuristics could be eliminated altogether using the fixed search space described above.

D. SUMMARY

It is clear from our tests that the heuristic search method of selecting sites is feasible. The current program does, however, have some limitations, and more work is necessary to make the method practical. This could be the topic of future theses. We mentioned some additional problems in survey planning which could also be solved by a heuristic search program. Finally, we discussed three possible modifications which could be made to our program.

APPENDIX A

THE PRIMARY DATA SETS

The six primary data sets are fictitious areas constructed to test the program performance in idealized scenarios. They are COAST, BAY, RIVER, ISLES, POINT, and MIXED, and represent an open coastline, a bay, a river, a group of islands, a point, and a mixture of coastline and islands. These six data sets were each tested under three different cases:

1. The range of the equipment is very long with respect to the dimensions of the survey and there is no existing geodetic control in the area.
2. The range of the equipment is long and there are three existing geodetic control stations in the area.
3. The range of the equipment is very restricted and there is no existing geodetic control in the area.

Figure 12 through Figure 29 display all eighteen scenarios, including the program solutions and the manual solutions.

FICTITIOUS COAST

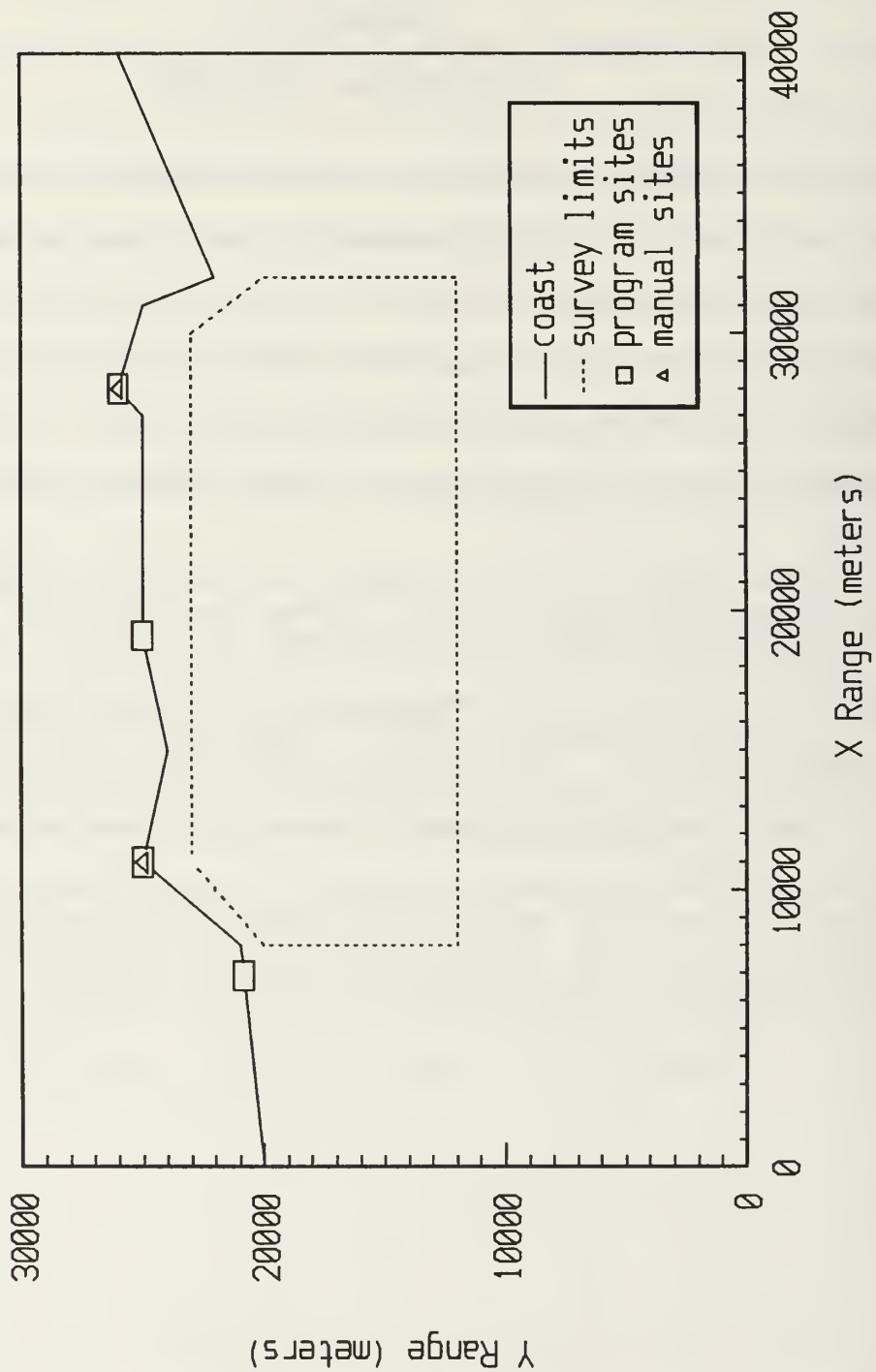


Figure 12. COAST Data Set Case 1.

FICTITIOUS COAST

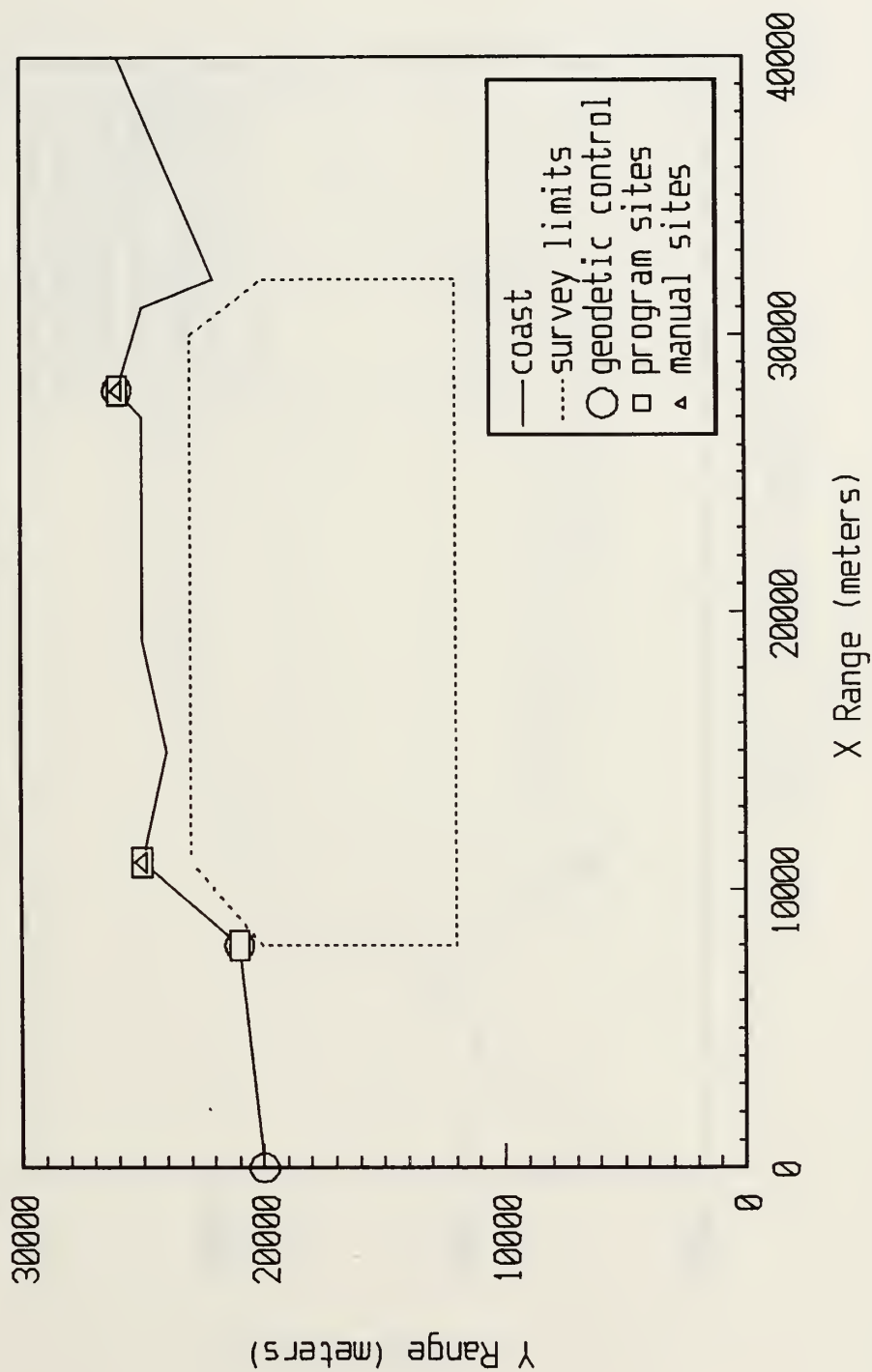


Figure 13. COAST Data Set Case 2.

FICTITIOUS COAST

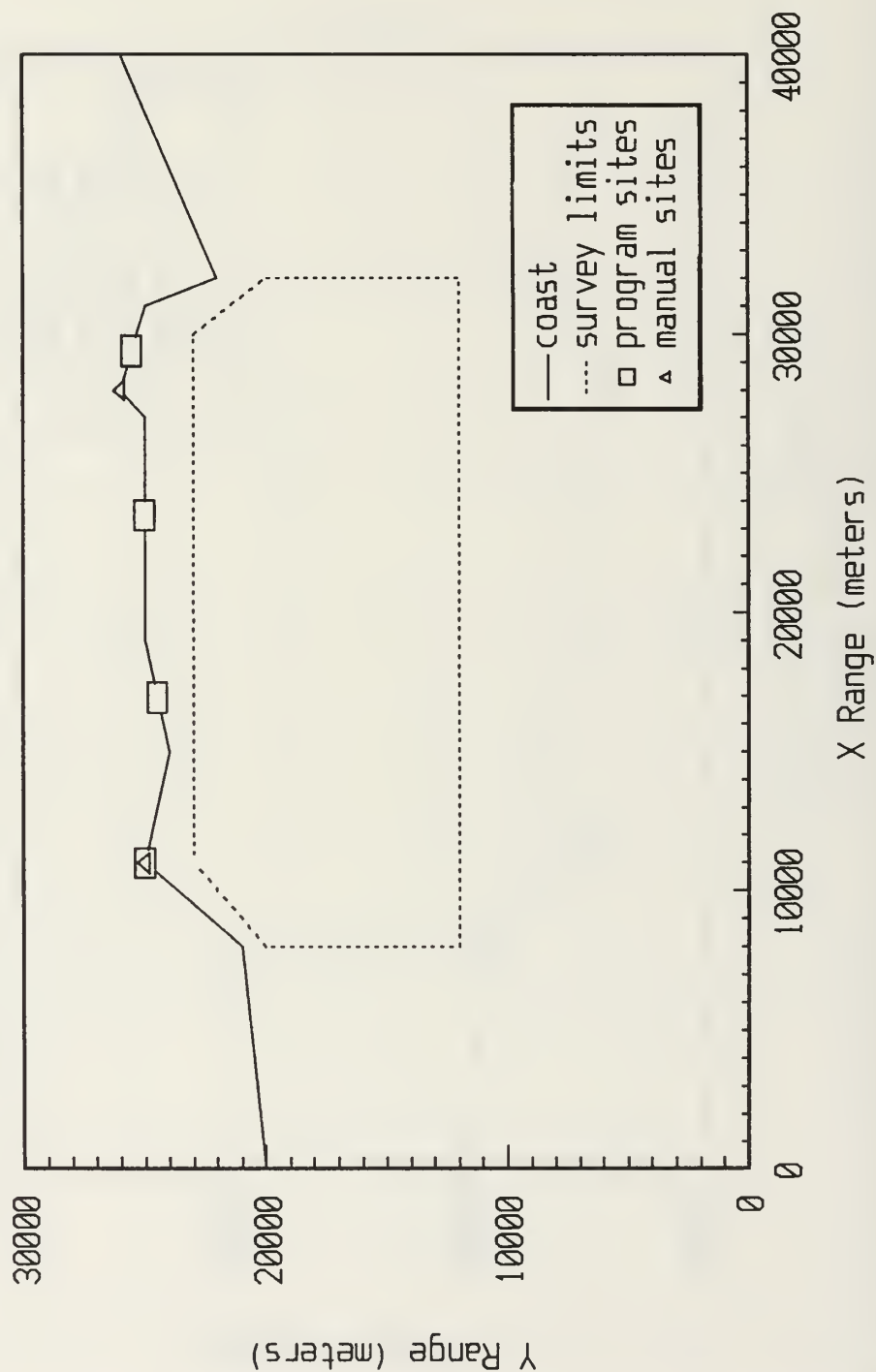


Figure 14. COAST Data Set Case 3.

FICTITIOUS BAY

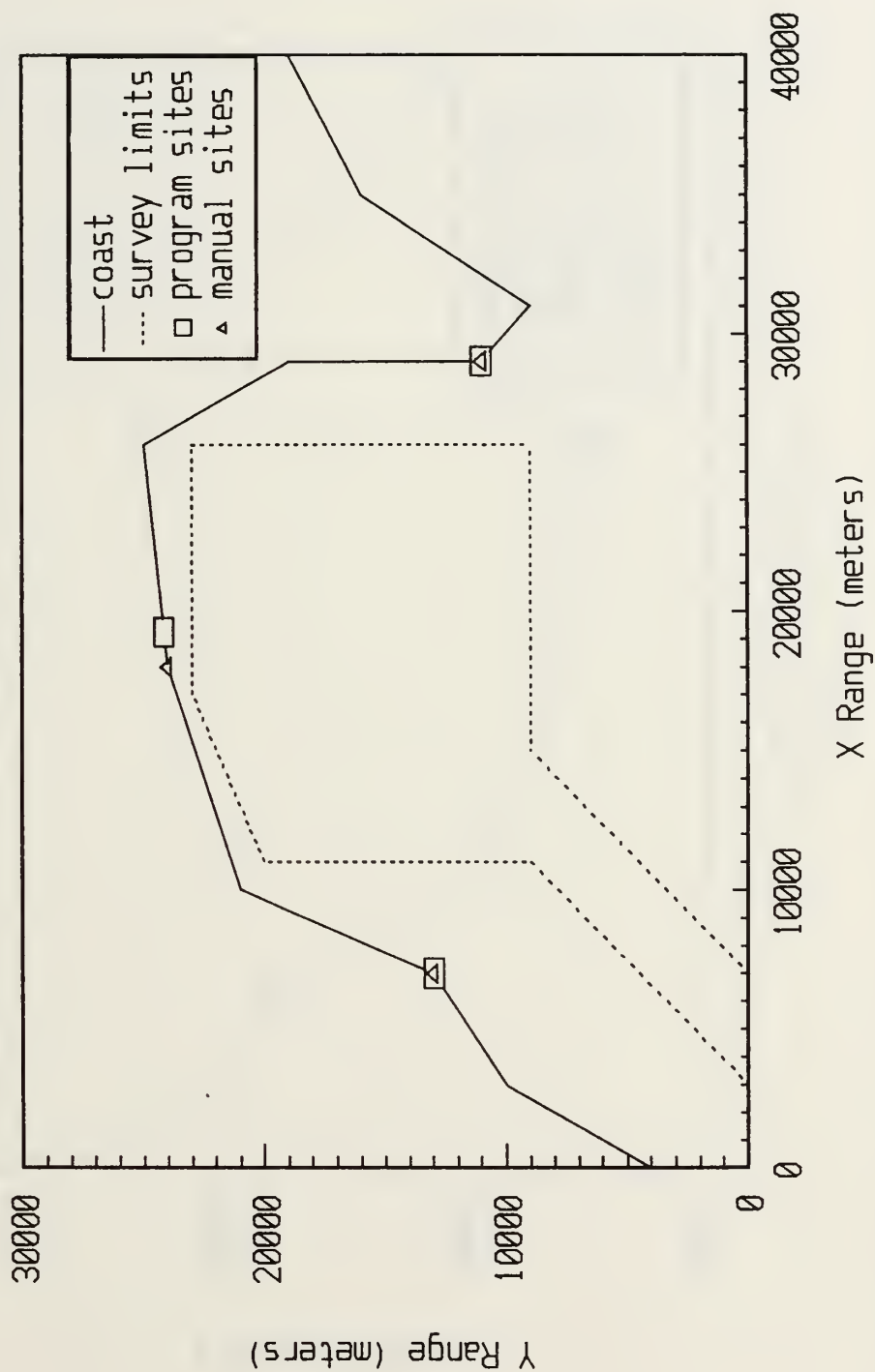


Figure 15. BAY Data Set Case 1.

FICTITIOUS BAY

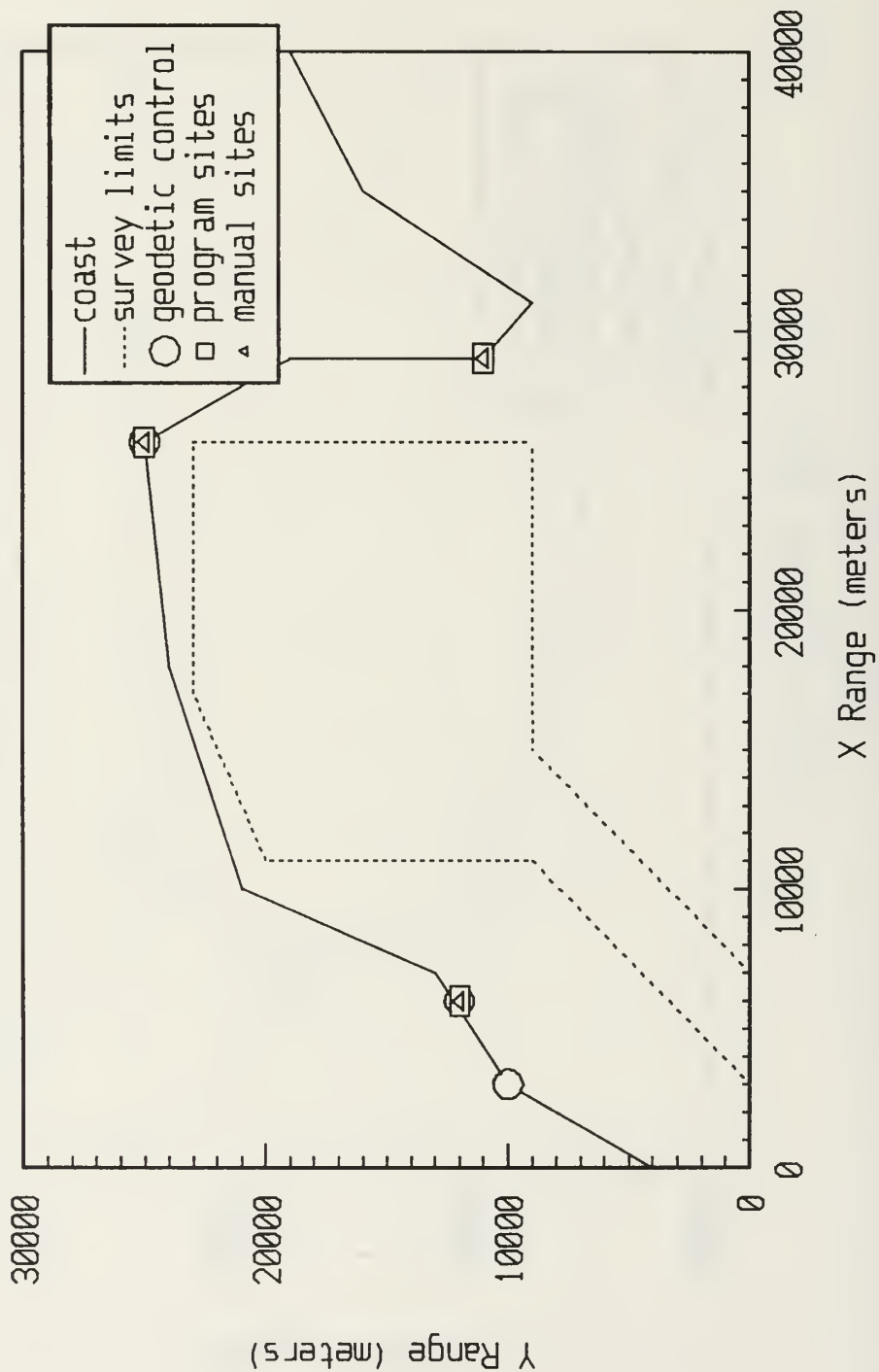


Figure 16. BAY Data Set Case 2.

FICTITIOUS BAY

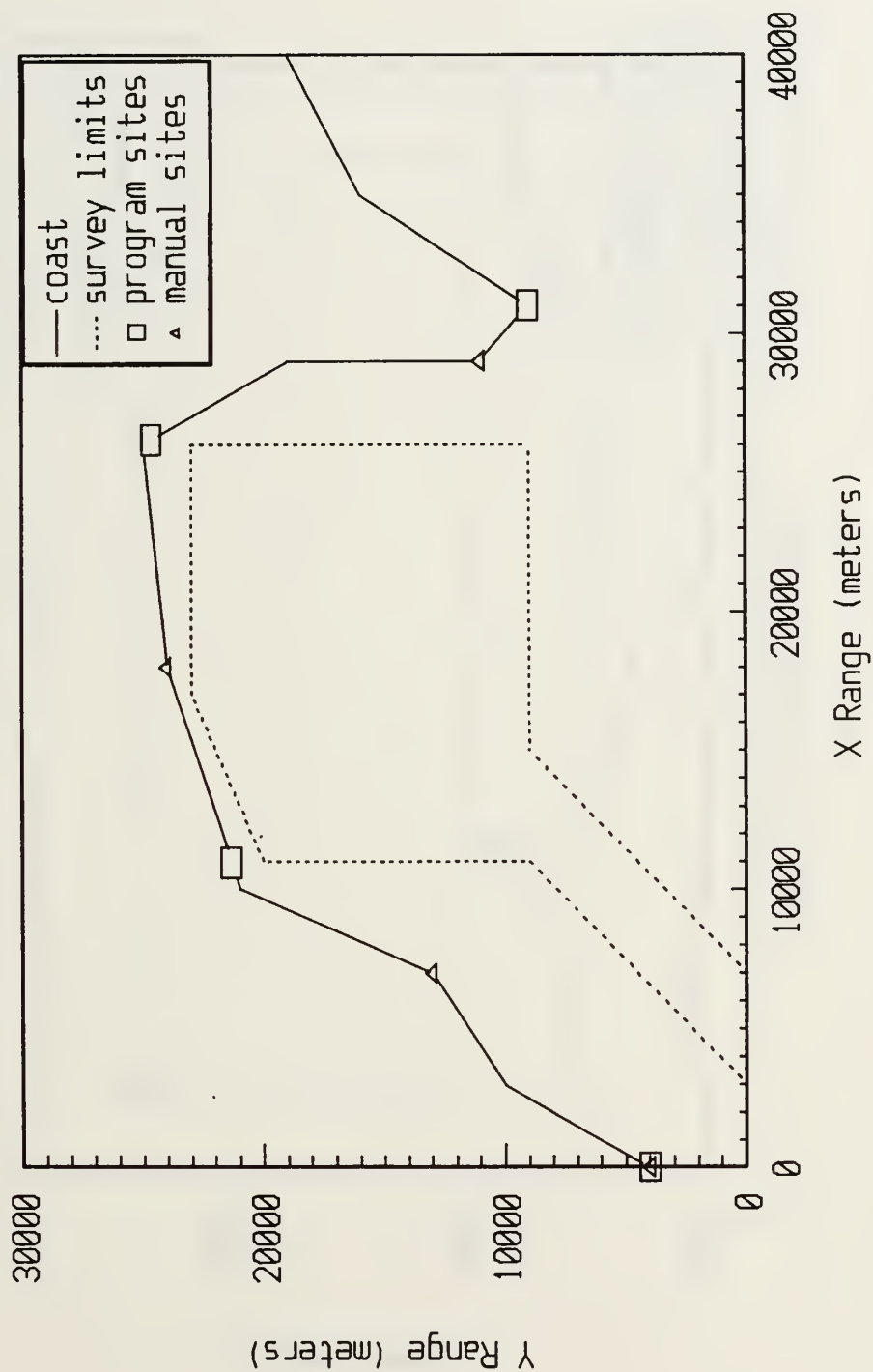


Figure 17. BAY Data Set Case 3.

FICTITIOUS RIVER

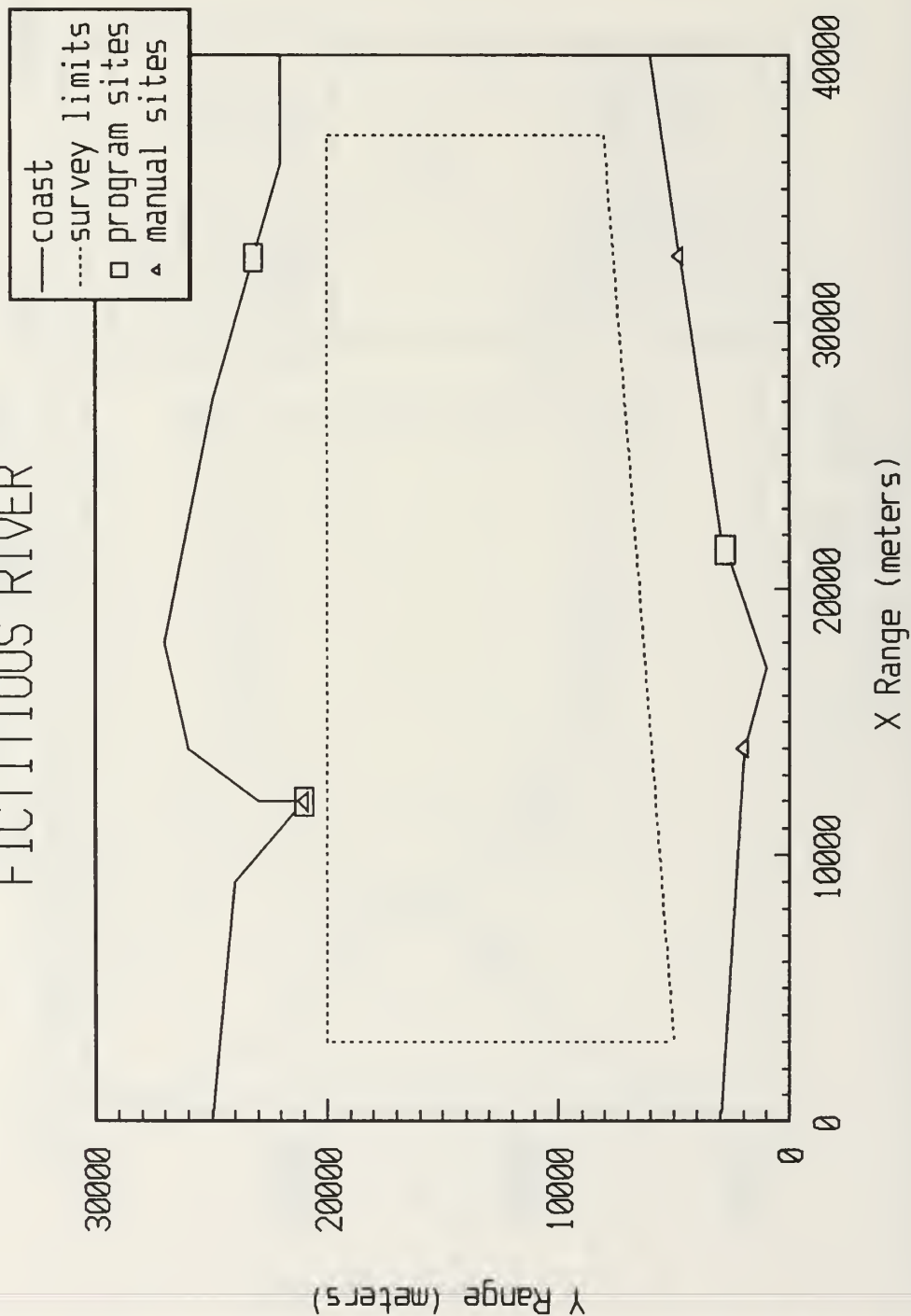


Figure 18. RIVER Data Set Case 1.

FICTITIOUS RIVER

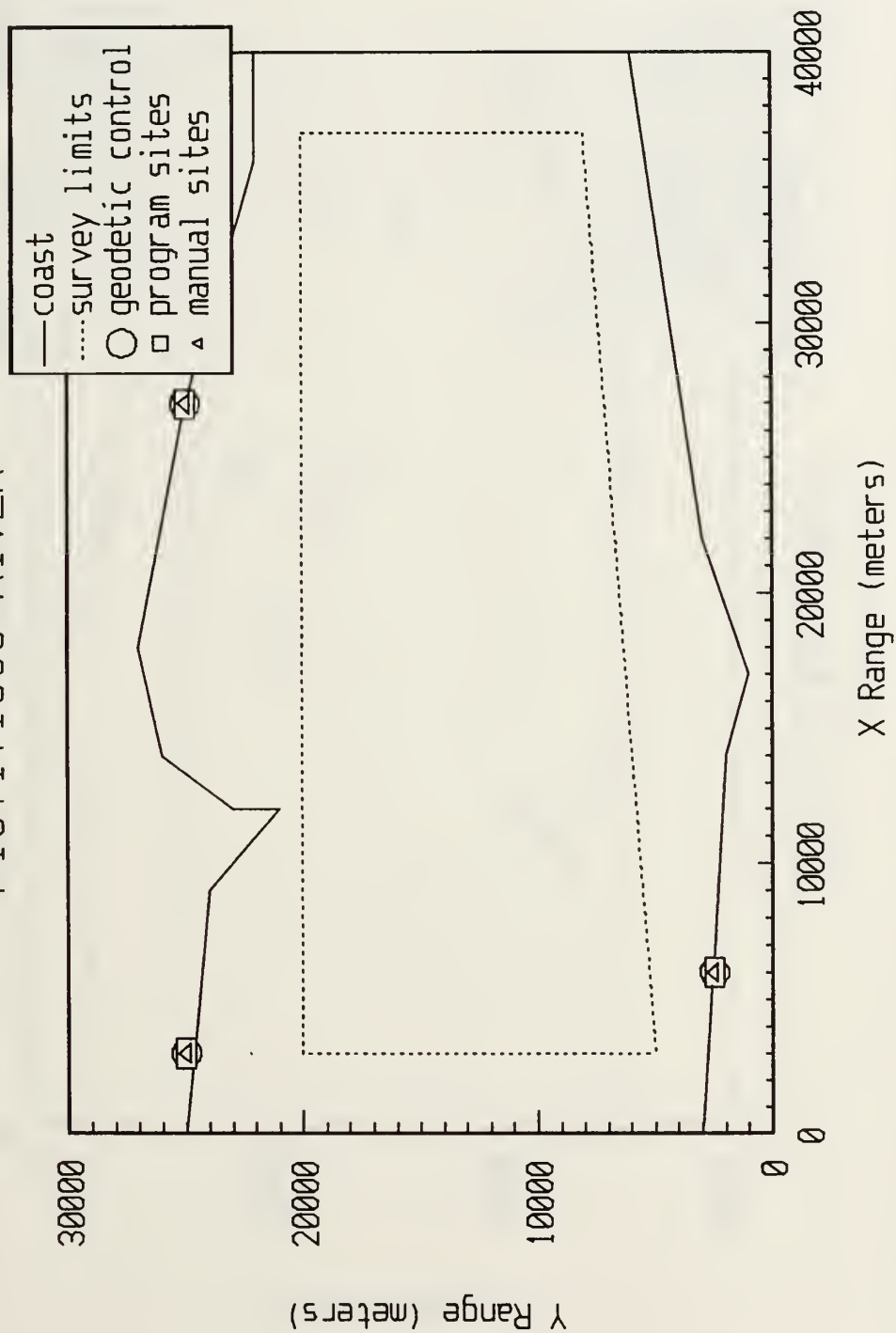


Figure 19. RIVER Data Set Case 2.

FICTITIOUS RIVER

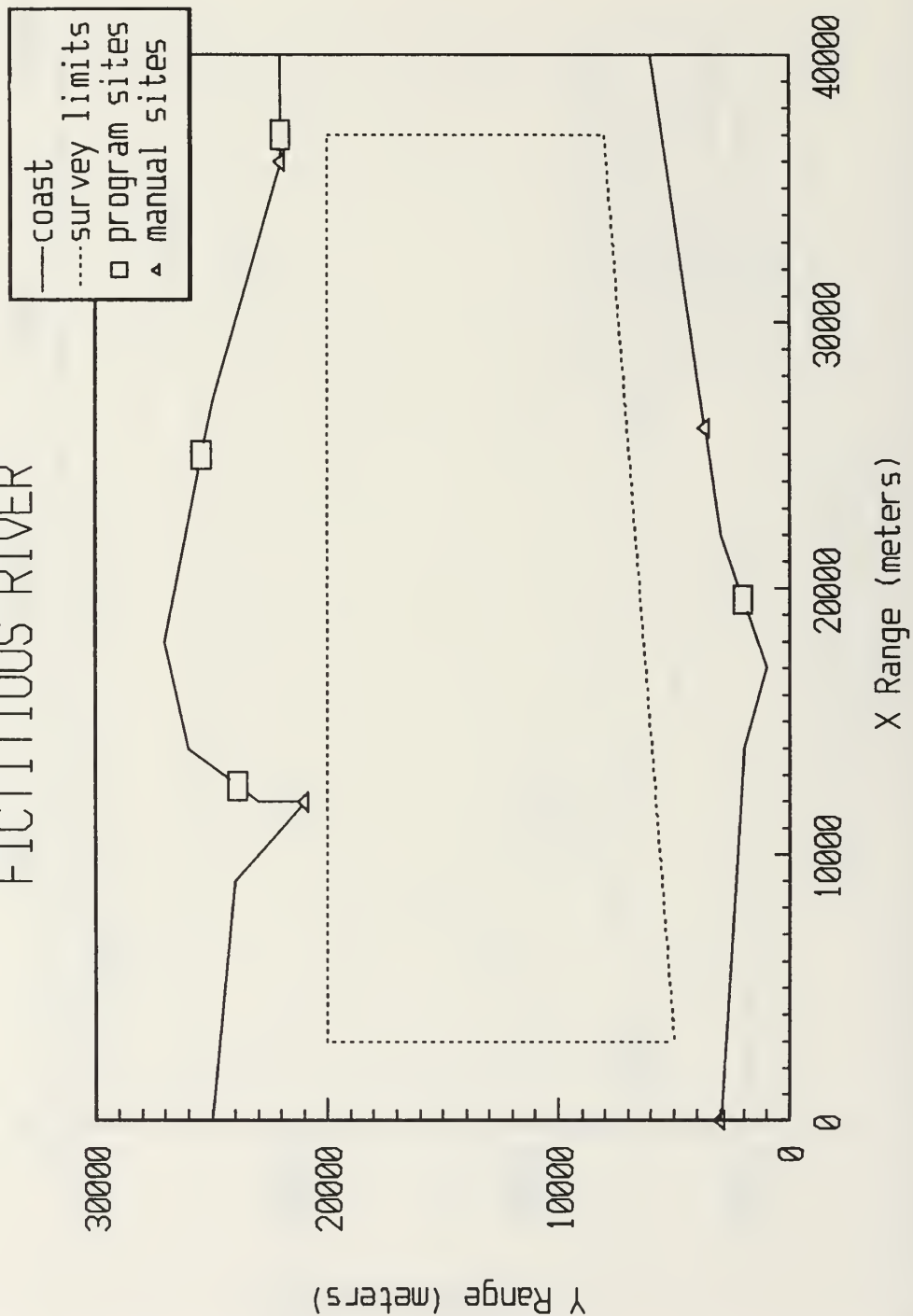


Figure 20. RIVER Data Set Case 3.

FICTITIOUS ISLES

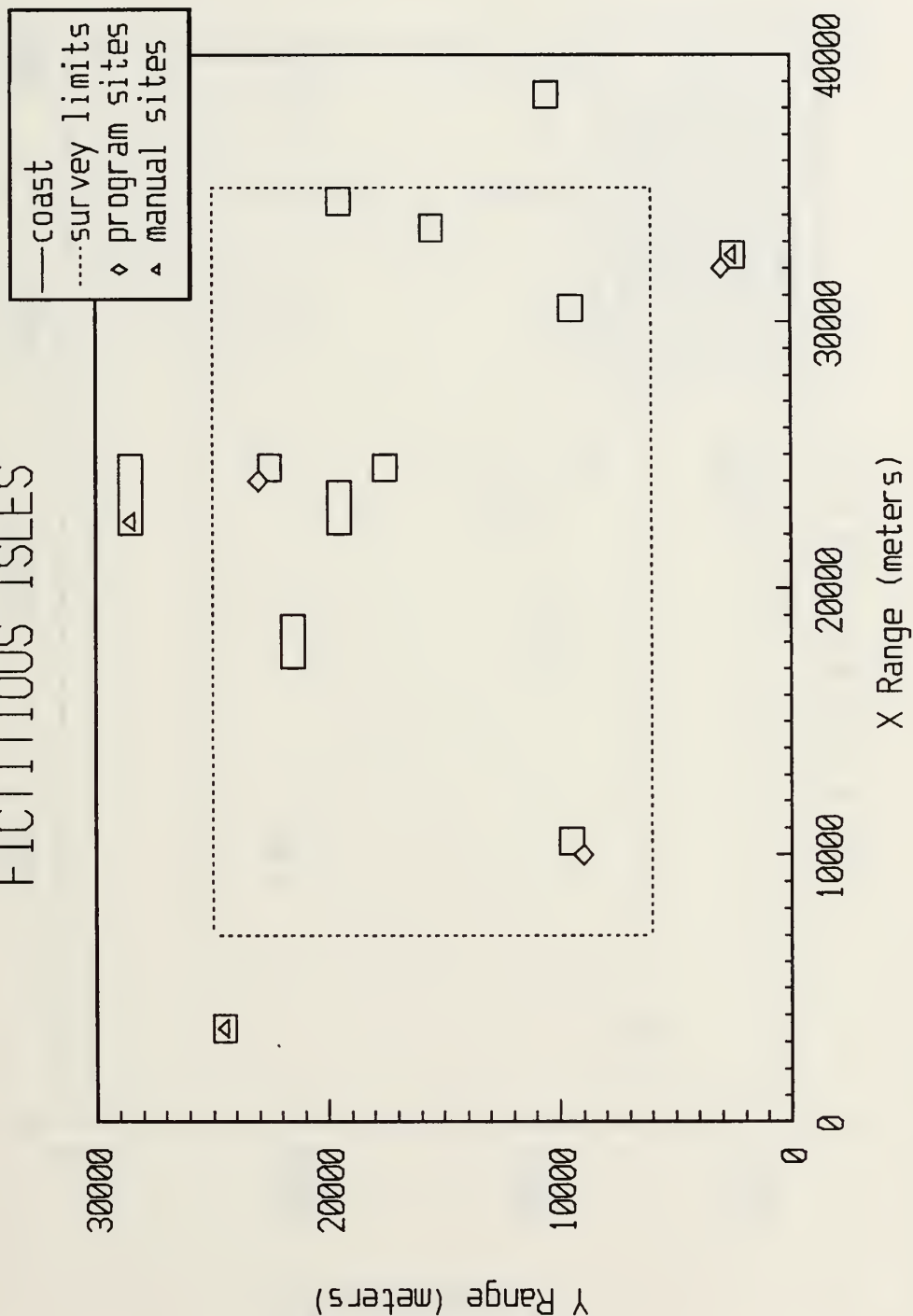
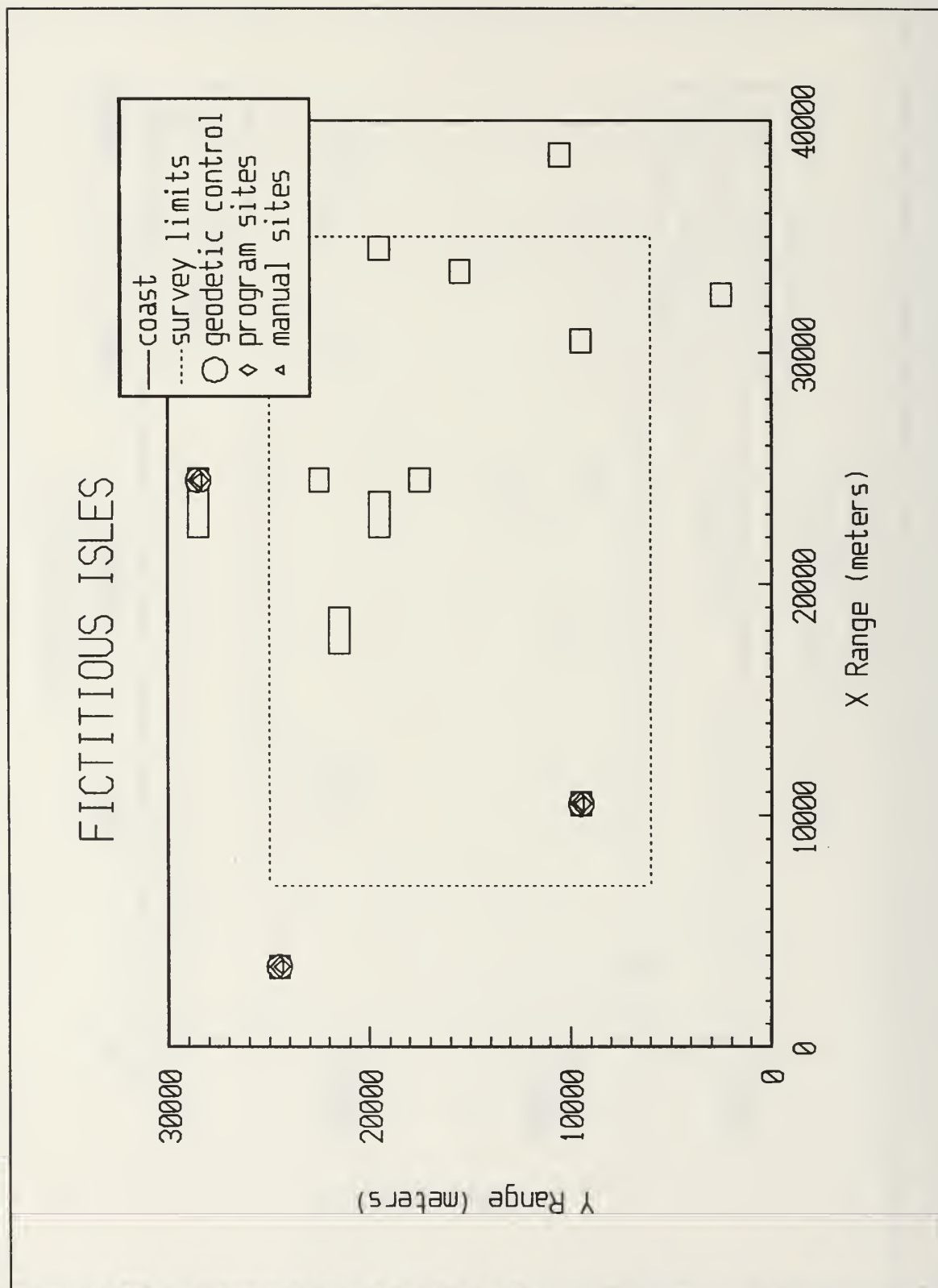


Figure 21. ISLES Data Set Case 1.



FICTITIOUS ISLES

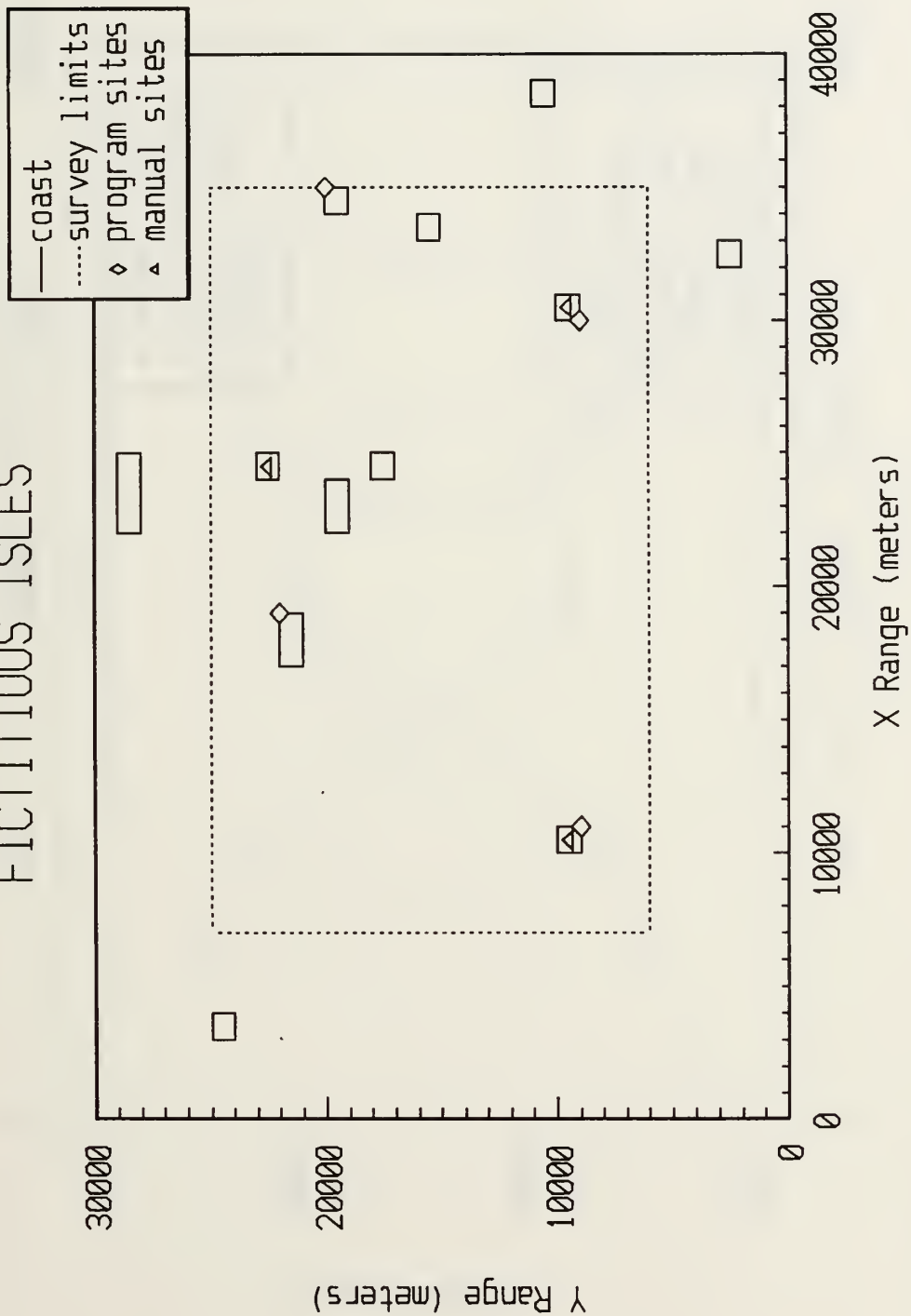


Figure 23. ISLES Data Set Case 3.

FICTITIOUS POINT

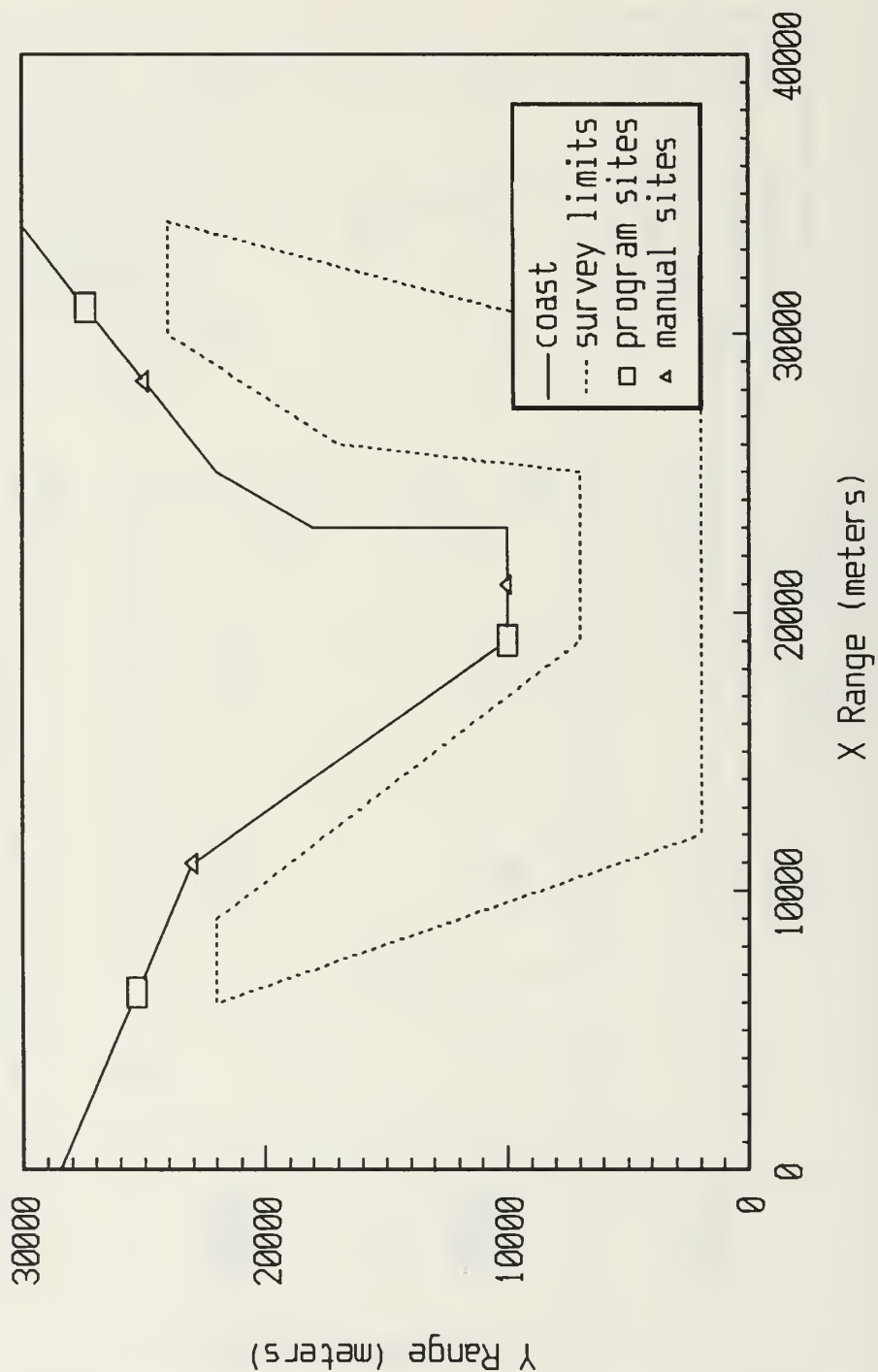


Figure 24. POINT Data Set Case 1.

FICTITIOUS POINT

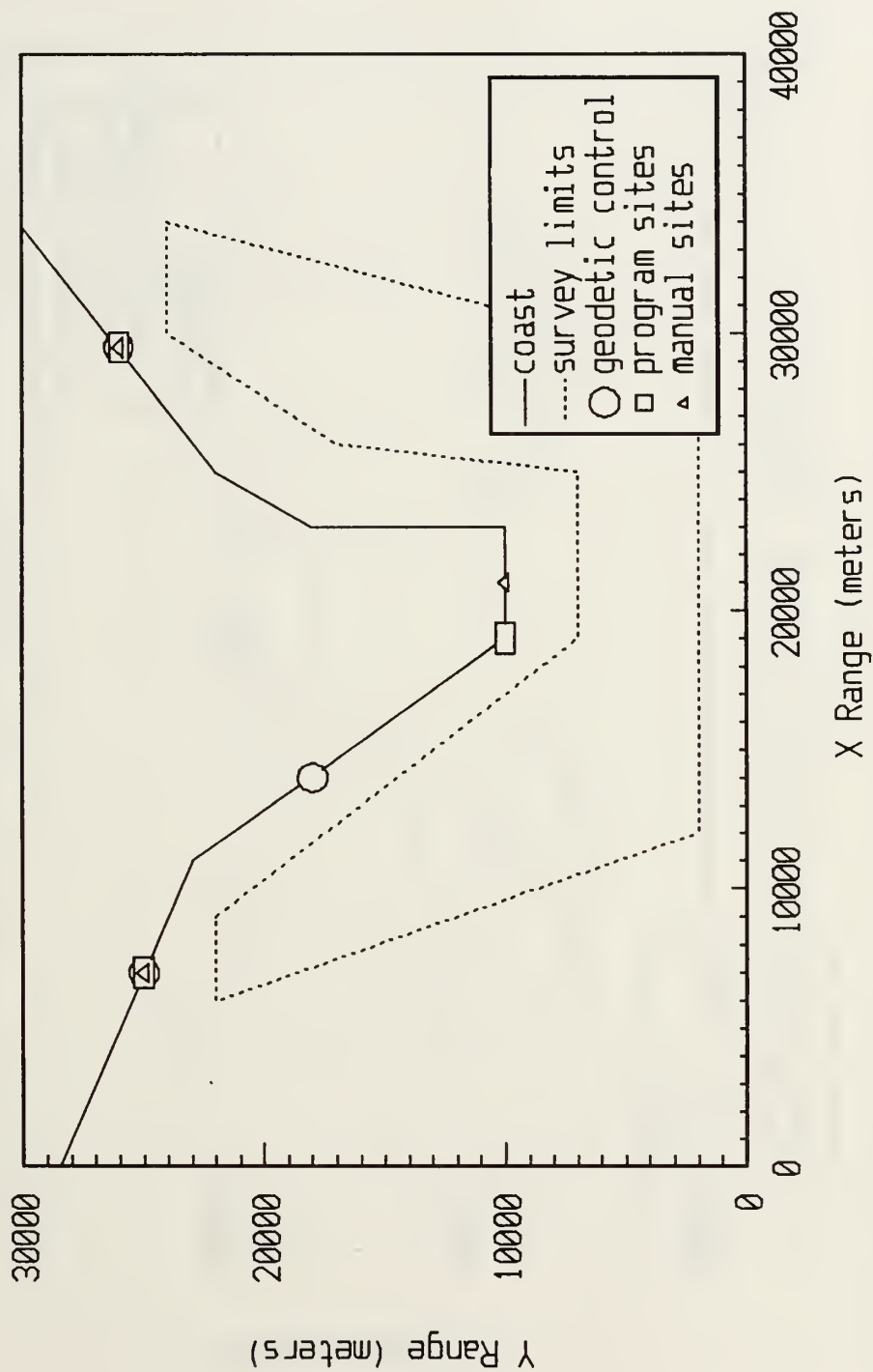


Figure 25. POINT Data Set Case 2.

FICTITIOUS POINT

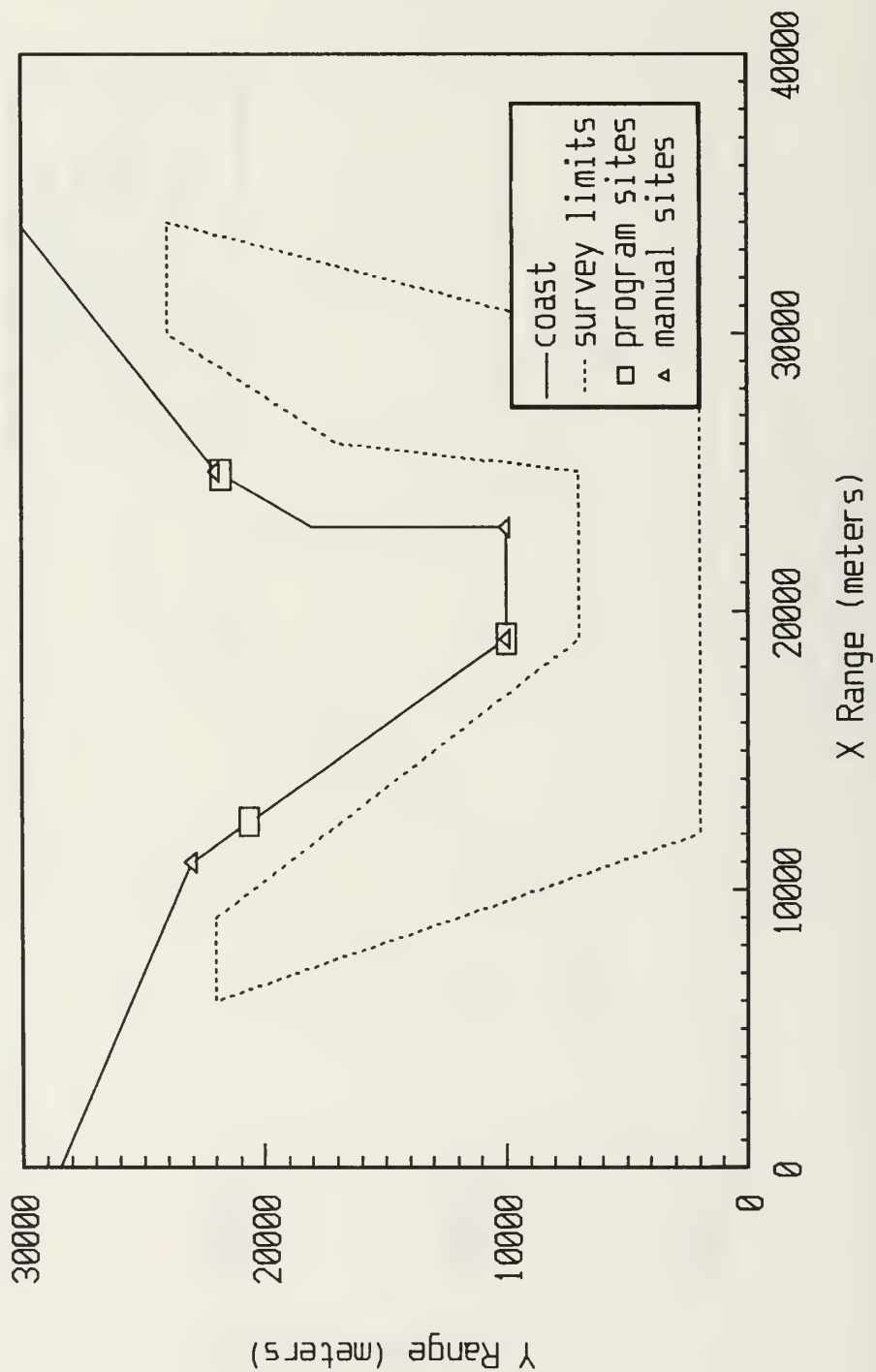


Figure 26. POINT Data Set Case 3.

FICTITIOUS BARRIER ISLANDS

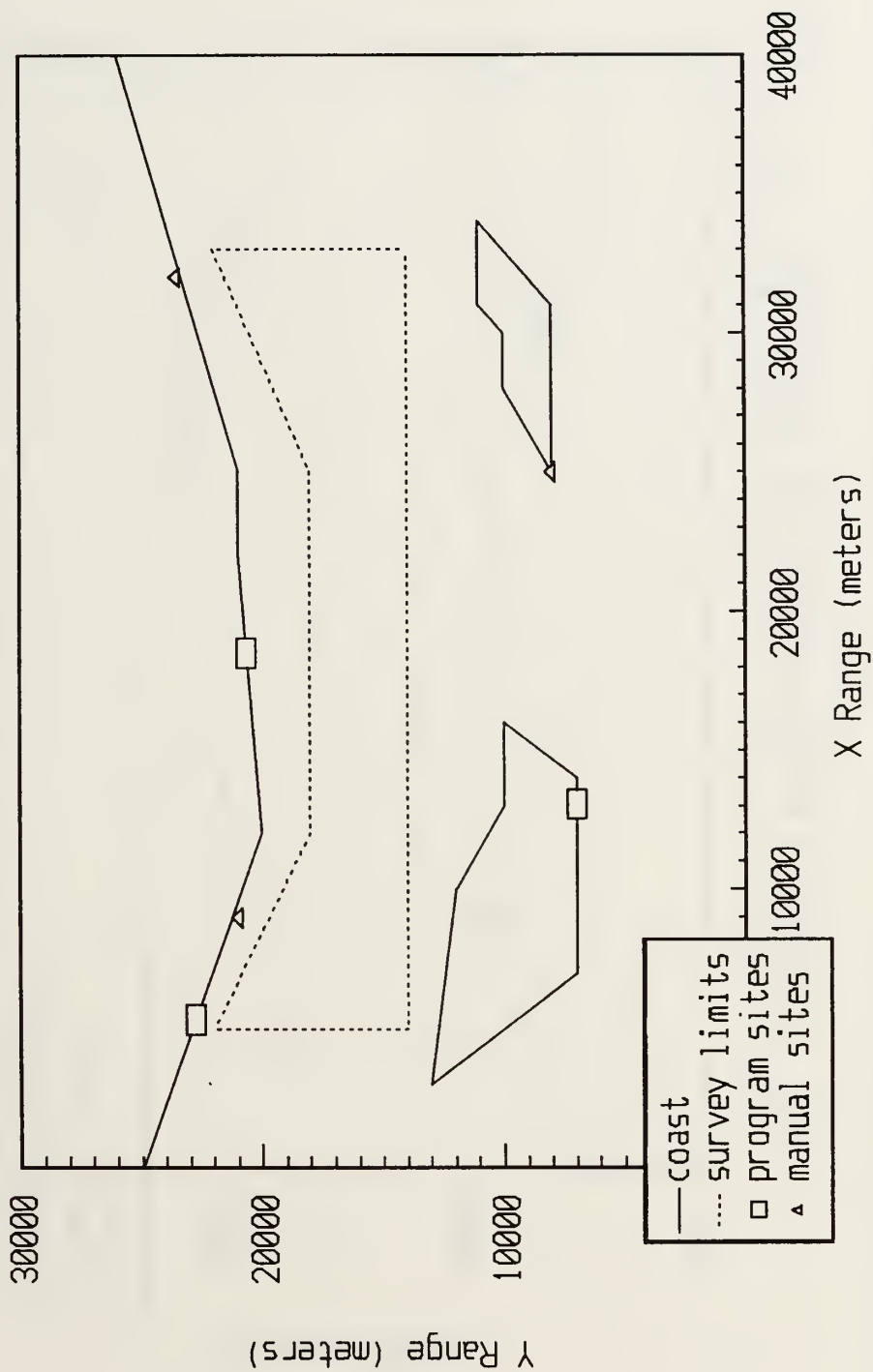


Figure 27. MIXED Data Set Case 1.

FICTITIOUS BARRIER ISLANDS

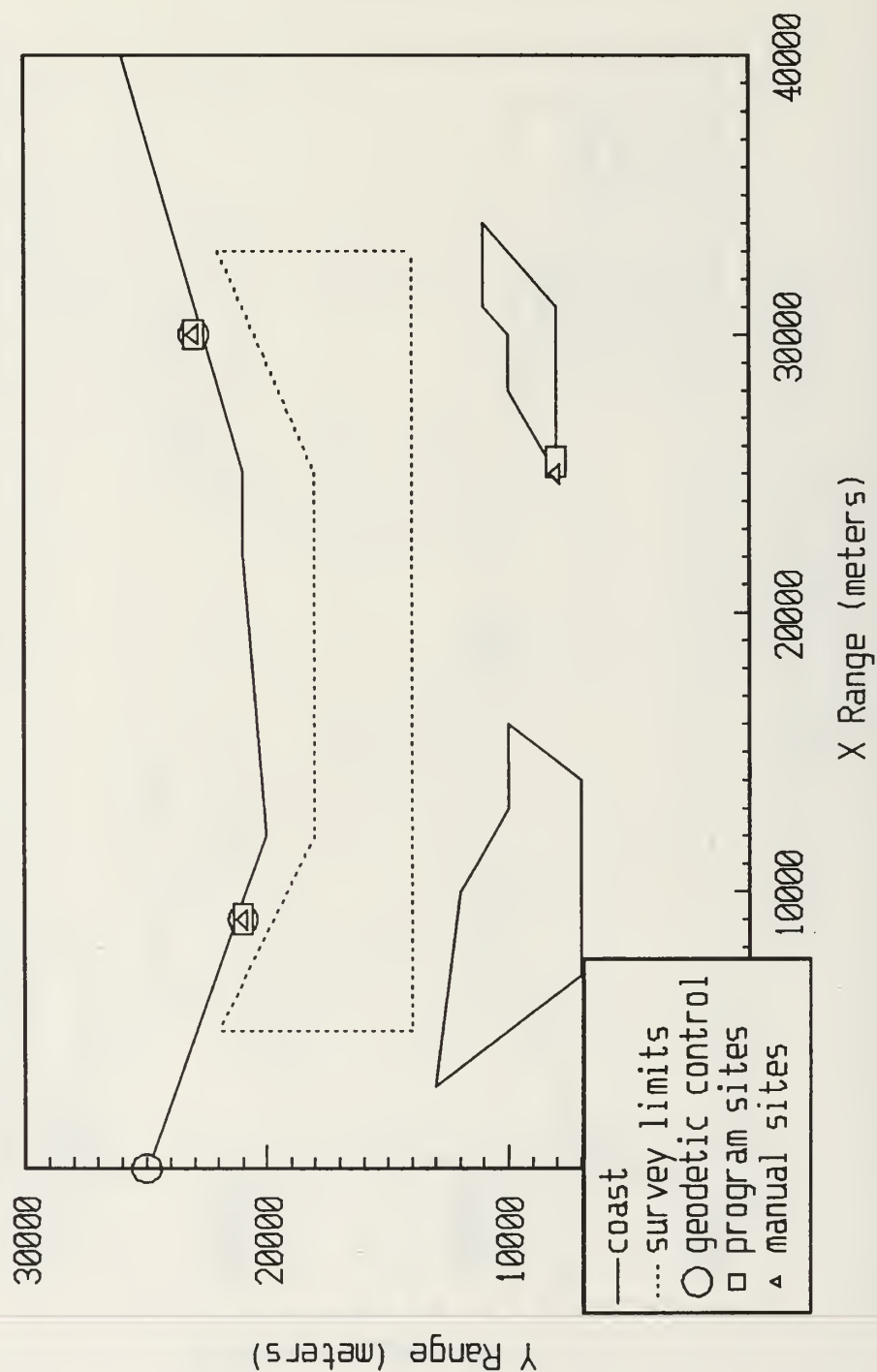


Figure 28. MIXED Data Set Case 2.

FICTITIOUS BARRIER ISLANDS

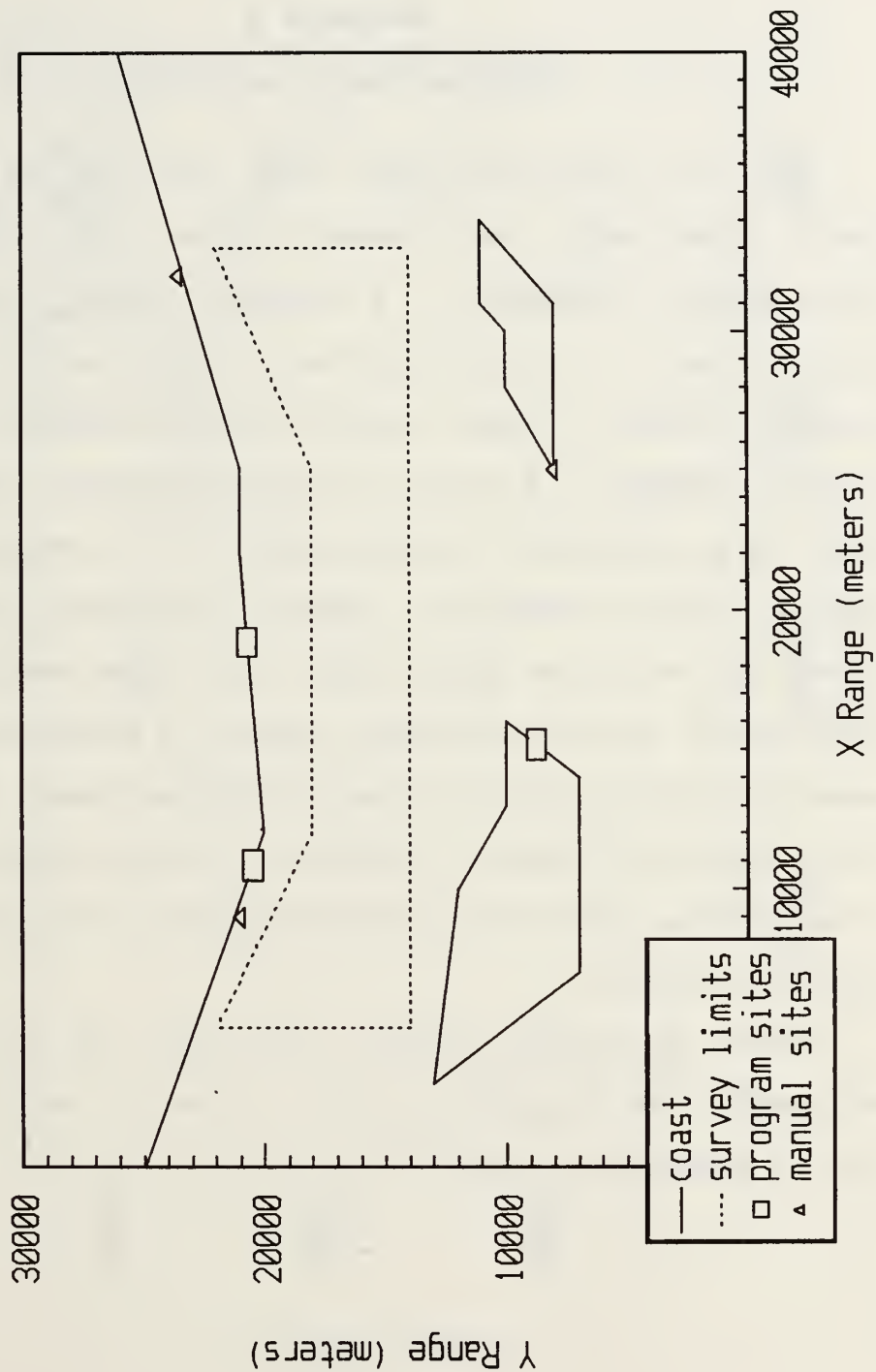


Figure 29. MIXED Data Set Case 3.

APPENDIX B

THE SECONDARY DATA SETS

The five secondary data sets were taken from actual charts, and each was tested under its own specific requirements. MONBAY is a coastal survey of the entire Monterey Bay. In this survey, medium-range positioning equipment (ARGO) is used, and no existing geodetic control is assumed. MONHAR is a survey of Monterey Harbor conducted with short-range equipment (Miniranger) with several existing geodetic control stations. OBISPO is a harbor and coastline survey of San Luis Obispo Bay with short-range equipment (Miniranger) and no existing control. SUISUN is a channel survey in Suisun Bay with short-range equipment (Miniranger) and no control. Finally, HELENA is a survey around the Island of Saint Helena with short-range equipment (Trisponder) and no existing control.

Figure 30 through Figure 34 display the five secondary data sets, with program solutions and manual solutions for all but the HELENA data set.

MONTEREY BAY

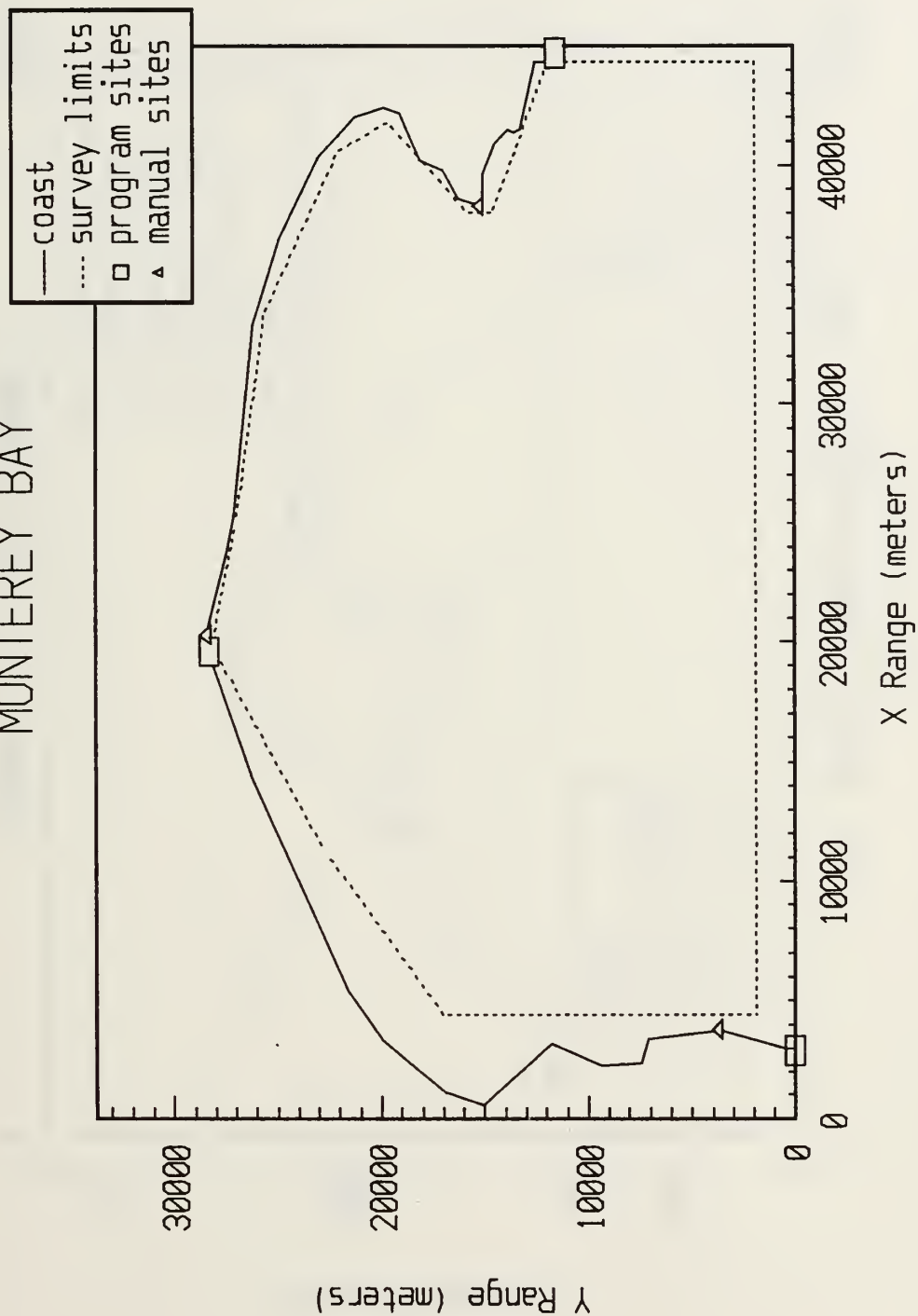


Figure 30. MONBAY Data Set.

MONTEREY HARBOR

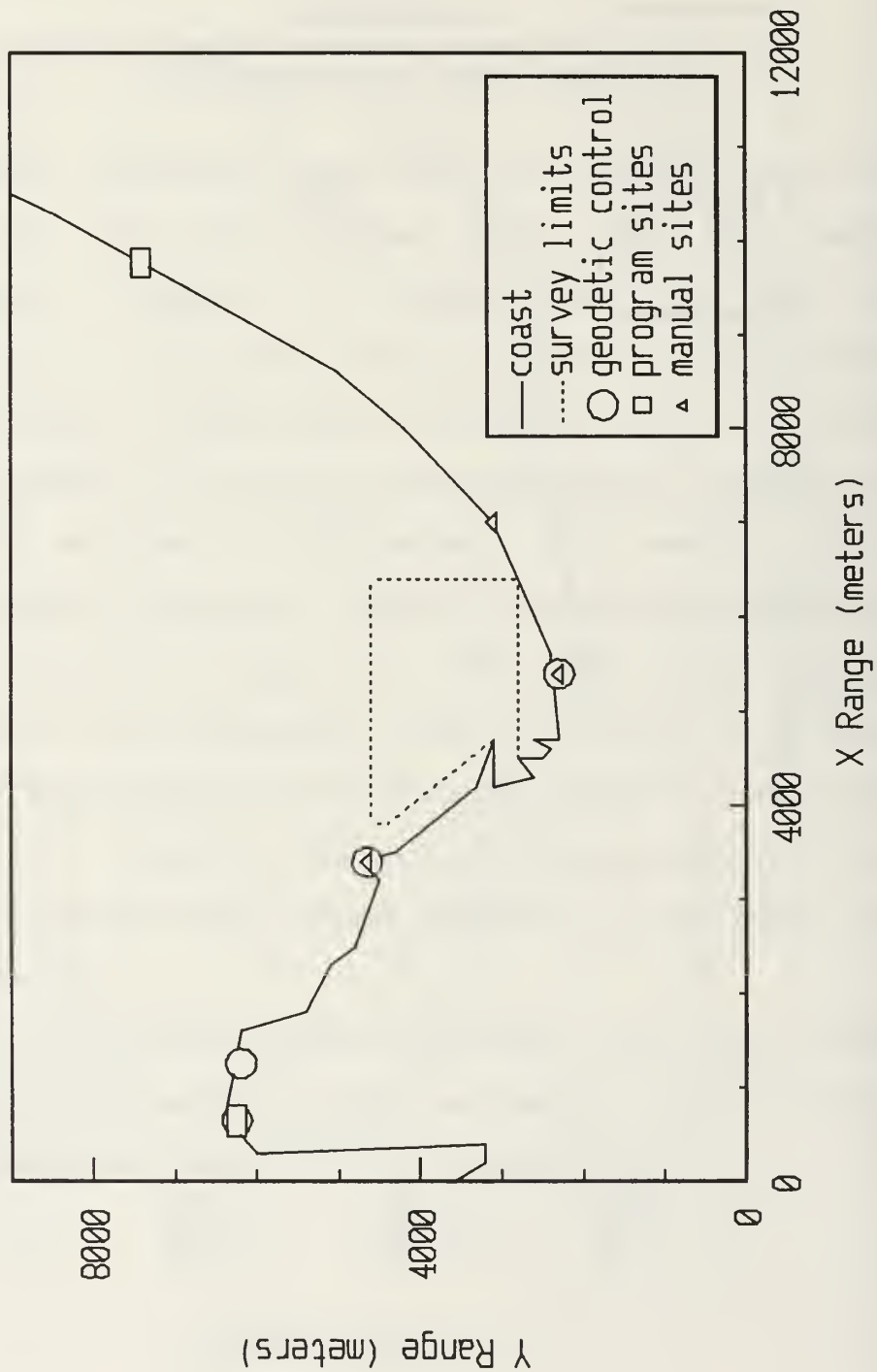


Figure 31. MONHAR Data Set.

SAN LUIS OBISPO BAY

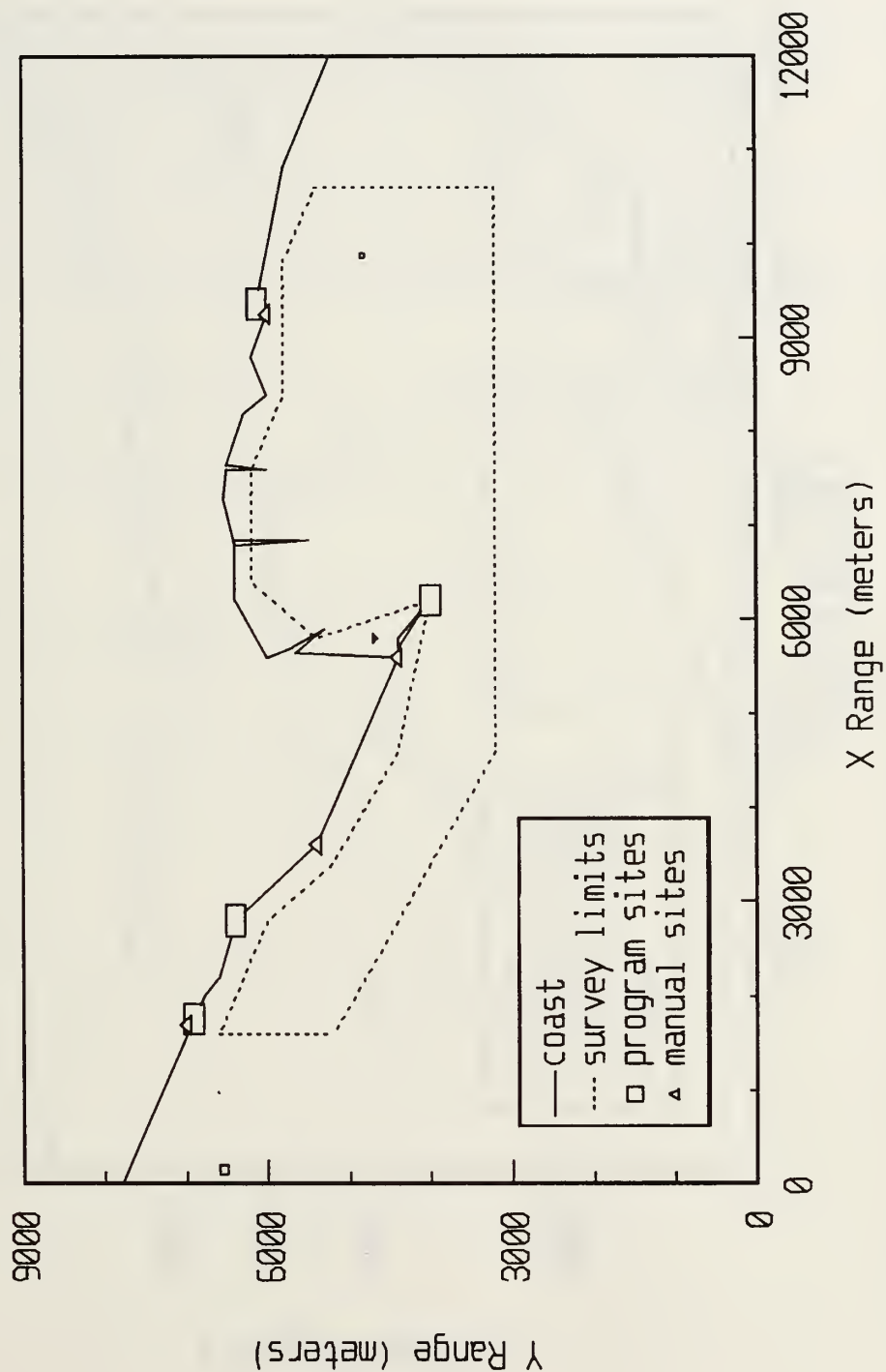


Figure 32. OBISPO Data Set.

SUISUN BAY

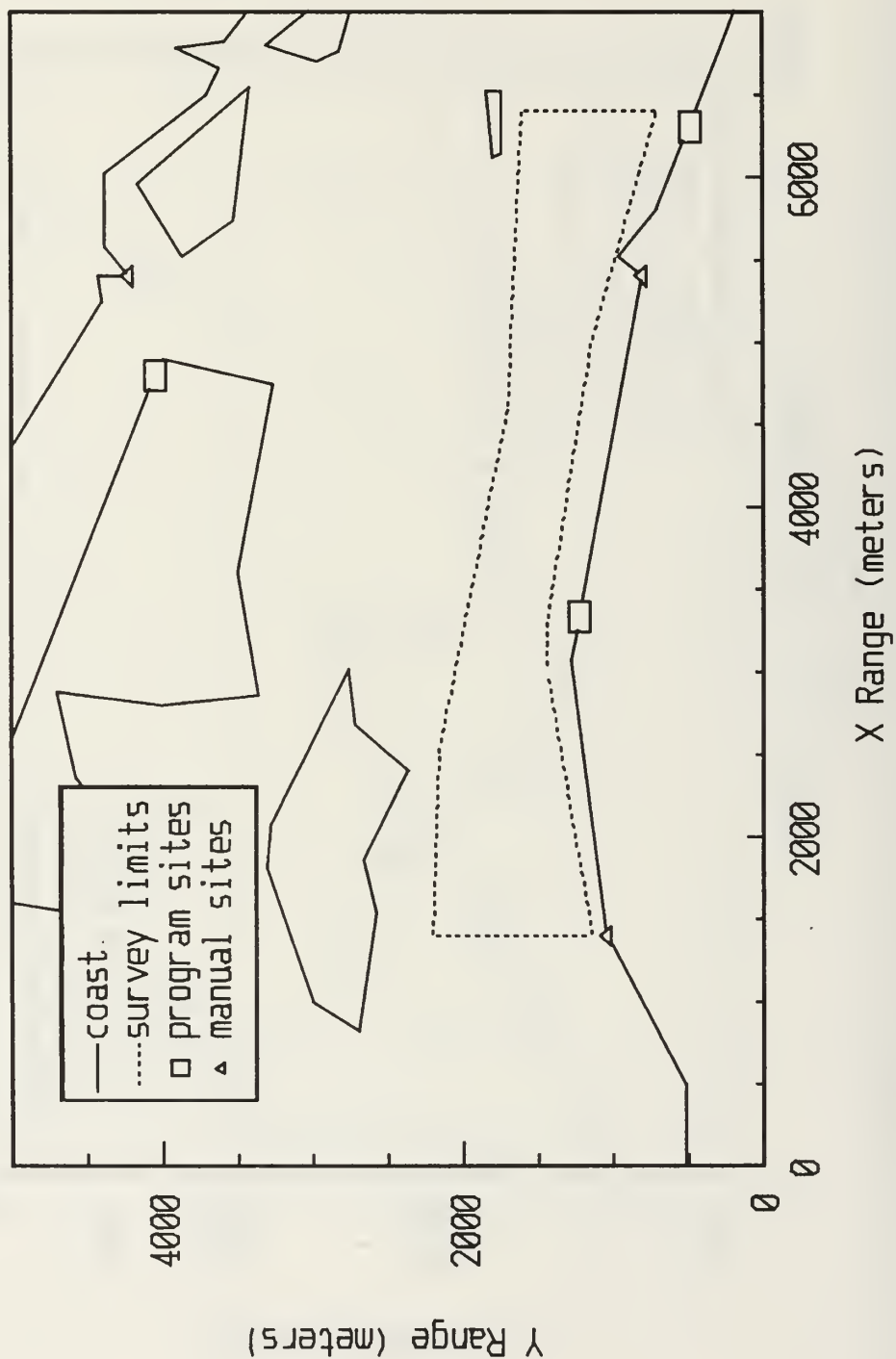


Figure 33. SUISUN Data Set.

ISLAND OF SAINT HELENA

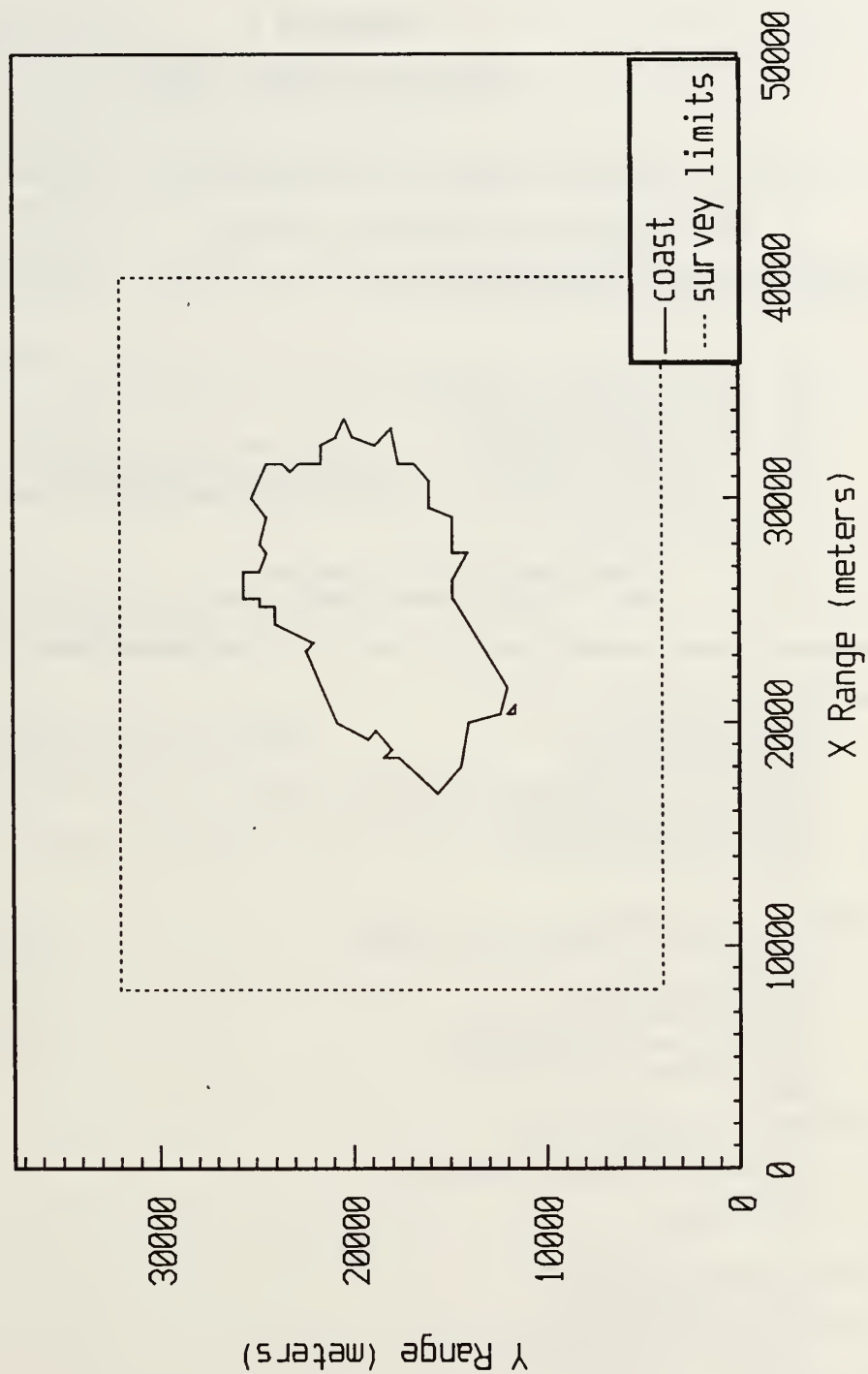


Figure 34. HELENA Data Set.

APPENDIX C

PROGRAM SOURCE CODE

/*=====

MODIFIED HEURISTIC SEARCH ALGORITHM

Original Program by Neil Rowe, 1988
Modified by Arnold Steed, May 1991

For an application, you must define 5 predicates:
(1) successor(State,Newstate)
(2) goal_reached(State)
(3) eval(State,Evaluation) -- estimates cost to goal
(4) cost(State,Cost) -- computes cost of a state
(5) a top-level predicate that initializes things and calls search

Note: "cost" must be non-negative
"eval" should be a lower bound

=====*/

```
search(Start,State):-
    clean_database,
    add_state(Start),
    repeat_if_agenda,
    pick_best_state(State),
    add_successors(State),
    retract(agenda(State,_,_)),
    measure_work.

pick_best_state(State):-
    asserta(best_state(dummy,dummy)),
    agenda(S,_,D),
    best_state(S2,D2),
    special_less_than(D,D2),
    retract(best_state(S2,D2)),
    asserta(best_state(S,D)),
    fail.

pick_best_state(State):-
    best_state(State,D),
    retract(best_state(State,D)),
    expunge,
    not(D=dummy), !.

add_successors(State):-
    goal_reached(State), !.
add_successors(State):-
    successor(State,Newstate),
    add_state(Newstate),
    fail.
add_successors(State):-
    retract(agenda(State,C,D)),
    asserta(used_state(State,C)),
```

```

expunge,
fail.

add_state(Newstate):-
    cost(Newstate,Cnew), !,
    eval(Newstate,Enew),
    D is Enew + Cnew,
    asserta(agenda(Newstate,Cnew,D)), !.
add_state(Newstate):-
    not(cost(Newstate,Cnew)),
    write('Warning: Your cost function failed on state '),
    write(Newstate), nl, !.
add_state(Newstate):-
    not(eval(Newstate,Enew)),
    write('Warning: Your evaluation function failed on state '),
    write(Newstate), nl, !.

repeat_if_agenda.
repeat_if_agenda:-
    agenda(,_,_),
    repeat_if_agenda.

special_less_than(X,dummy):- !.
special_less_than(X,Y):-
    X < Y.

clean_database:-
    abolish( agenda/3 ),
    abolish( used_state/2 ),
    abolish( best_state/2 ).

measure_work:-
    count_up(agenda(X,C,D),NA),
    count_up(used_state(S,C),NB),
    write(NA),
    write(' incompletely examined state(s) and '),
    write(NB),
    write(' examined state(s)'), nl, !.

count_up(P,_):-
    ctr_set(0,0),
    call( P ),
    ctr_inc(0,_),
    fail.
count_up(,N):-
    ctr_is(0,N).

```

```

/*=====

SURVEY PLANNING PREDICATES

Original Program by Arnold Steed, May 1991

Note: Input data files must have the extension ".dat"
and contain the following prolog predicates:

(1) chart_limits( [Lower_left,Upper_right] )
(2) shore_line( Polygon_and_fragment_list )
(3) survey_limits( Polygon_list )
(4) geodetic_control( Station_list )
(5) range_limits( Minimum, Maximum )
(6) site_cost( Site, Cost )
(7) site_deviation( Floating_point_number )

=====*/

plan( Survey_file, Site_list ):-
    clean_up,
    initialize_data( Survey_file, Area ),
    search( [[],_,Area], [Site_list|_] ).

clean_up:-
    abolish( range_constraints/0 ),
    abolish( shore_segments/1 ), !.

initialize_data( File, Area ):-
    name( File, List1 ),
    append( List1, ".dat", List2 ),
    name( Data_file, List2 ),
    reconsult( Data_file ),
    chart_limits( Chart ),
    shore_line( Shore ),
    trim_shore( Chart, Shore, Trimmed_shore ),
    asserta( shore_segments( Trimmed_shore ) ),
    survey_limits( Area ),
    check_range_constraints( Area ).

trim_shore( [[X0,Y0],[X1,Y1]], Shore, Trimmed_shore ):-
    make_segment_list( [[X0,Y0],[X0,Y1],[X1,Y1],[X1,Y0],[X0,Y0]],
        Chart ),
    make_segment_list( Shore, Seg_shore ),
    trim_outside( Chart, Seg_shore, Trimmed_shore ), !.

check_range_constraints( Area ):-
    extreme_distance( Area, Distance ),
    range_limits( _, Max ),
    ifthen( Distance > Max/2,
        assert(range_constraints) ).

extreme_distance( Area, Distance ):-
    compress_list_of_lists( Area, List ),
    relative_distances( List, _, Distance ).

/*-----
Define Goal
-----*/

goal_reached( [_,_,[]] ).

```

```

/*-----
                        Define Successors
-----*/

successor( [_,_,[]], _ ):-
    !, fail.
successor( [[],_,Area], [Site_pair,C,New_area] ):-
    two_initial_sites( Area, Site_pair ),
    find_area_covered( Site_pair, Area, New_area, A ),
    C is A/2.
successor( [[],_,Area], [[Site],0.0,Area] ):-
    initial_site( Area, Site ).
successor( [[Site|List],C,Area], [[New_site,Site|List],New_C,New_area] ):-
    range_constraints,
    special_site( Site, New_site ),
    valid_site( New_site, [Site|List] ),
    find_area_covered( [New_site,Site|List], Area, New_area, A ),
    length( [Site|List], N ),
    Total_area_covered is C*N + A,
    New_C is Total_area_covered/(N + 1).
successor( [Site_list,C,Area], [[New_site|Site_list],New_C,New_area] ):-
    not( Site_list = [] ),
    center_of_area( Area, Center ),
    choose_one_site( Center, Site_list, New_site ),
    find_area_covered( [New_site|Site_list], Area, New_area, A ),
    length( Site_list, N ),
    Total_area_covered is C*N + A,
    New_C is Total_area_covered/(N + 1).

two_initial_sites( Area, Pair ):-
    center_of_area( Area, P0 ),
    geodetic_control( Sites ),
    find_best_pair( P0, Sites, Pair ).
two_initial_sites( Area, Pair ):-
    center_of_area( Area, P0 ),
    get_radial_intersections( P0, Intersections ),
    find_best_pair( P0, Intersections, Pair ).

initial_site( _, Site ):-
    geodetic_control( Sites ),
    member( Site, Sites ).
initial_site( Area, Site ):-
    center_of_area( Area, Center ),
    most_distant_vertex( Area, Center, Vertex ),
    find_nearest_shore( Vertex, Site ).
initial_site( Area, Site ):-
    center_of_area( Area, Center ),
    make_segment_list( Area, Seg_list ),
    nearest_point_on_list( Seg_list, Center, Point ),
    find_nearest_shore( Point, Site ).

choose_one_site( Center, [Site], New_site ):-
    geodetic_control( [] ),
    new_site( Center, [Site], Site, [_,New_site] ).
choose_one_site( Center, Site_list, Old_site ):-
    geodetic_control( Control ),
    setof( Site,
        old_site( Center, Site_list, Control, Site ),
        [[_,Old_site]] ).
choose_one_site( Center, Site_list, New_site ):-
    member( Old_site, Site_list ),
    setof( Site,
        new_site( Center, Site_list, Old_site, Site ),

```

```

    [[_,New_site] | _] ).

old_site( Center, [Site|List], Control, [Q,Old_site] ):-
    member( Old_site, Control ),
    not( member(Old_site,List) ),
    pair_quality( Center, [Site,Old_site], Q ),
    valid_site( Old_site, [Site|List] ).

new_site( [CX,CY], Site_list, [SX,SY], [Q,Site] ):-
    X is CX - (CY - SY),
    Y is CY + (CX - SX),
    find_nearest_shore( [X,Y], Site ),
    valid_site( Site, Site_list ),
    pair_quality( [CX,CY], [Site,[SX,SY]], Q ).
new_site( [CX,CY], Site_list, [SX,SY], [Q,Site] ):-
    X is CX + (CY - SY),
    Y is CY - (CX - SX),
    find_nearest_shore( [X,Y], Site ),
    valid_site( Site, Site_list ),
    pair_quality( [CX,CY], [Site,[SX,SY]], Q ).
new_site( [CX,CY], Site_list, [SX,SY], [Q,Site] ):-
    X is SX + 1.5*(SY - CY),
    Y is SY - 1.5*(SX - CX),
    find_nearest_shore( [X,Y], Site ),
    valid_site( Site, Site_list ),
    pair_quality( [CX,CY], [Site,[SX,SY]], Q ).
new_site( [CX,CY], Site_list, [SX,SY], [Q,Site] ):-
    X is SX - 1.5*(SY - CY),
    Y is SY + 1.5*(SX - CX),
    find_nearest_shore( [X,Y], Site ),
    valid_site( Site, Site_list ),
    pair_quality( [CX,CY], [Site,[SX,SY]], Q ).

special_site( Site, New_site ):-
    range_limits( _, Max ),
    R is Max/2,
    shore_segments( List ),
    member( Segment, List ),
    seg_circle_intersect( Segment, [Site,R], New_site ).

find_best_pair( _, [P1,P2], [P1,P2] ):- !.
find_best_pair( P0, [P1|Points], Pair ):-
    find_best_pair( P0, Points, Pair1 ),
    find_best_pair2( P0, [P1|Points], Pair2 ),
    pair_quality( P0, Pair1, Q1 ),
    pair_quality( P0, Pair2, Q2 ),
    ifthenelse( Q1 < Q2, Pair = Pair1, Pair = Pair2 ), !.

find_best_pair2( _, [P1,P2], [P1,P2] ):- !.
find_best_pair2( P0, [P1,P2|Points], Pair ):-
    find_best_pair2( P0, [P1|Points], Pair1 ),
    pair_quality( P0, Pair1, Q1 ),
    pair_quality( P0, [P1,P2], Q2 ),
    ifthenelse( Q1 < Q2, Pair = Pair1, Pair = [P1,P2] ), !.

pair_quality( P0, [P1,P2], Q ):-
    angle_of_intersection( P1, P2, P0, A ),
    Q is (A - pi/2)^2.

find_nearest_shore( Point, Site ):-
    shore_segments( Shore ),
    nearest_point_on_list( Shore, Point, Site ), !.

```

```

/*-----
    Ensure state is not virtually identical to existing state
-----*/

valid_site( Site, Site_list ):-
    not( member( Site, Site_list ) ),
    sort( [Site|Site_list], Set ),
    agenda_check( Set ),
    used_state_check( Set ).

agenda_check( Set ):-
    agenda( [S,_,_],_,_ ),
    sort( S, Set2 ),
    same_set( Set, Set2 ), !,
    fail.
agenda_check( _ ).

used_state_check( Set ):-
    used_state( [S,_,_],_ ),
    sort( S, Set2 ),
    same_set( Set, Set2 ), !,
    fail.
used_state_check( _ ).

same_set( Set_1, Set_2 ):-
    site_deviation( C ),
    mean_square_error( points, Set_1, Set_2, E ),
    E < C.

/*-----
    Survey area coverage
-----*/

find_area_covered( Sites, Old_area, New_area, A ):-
    make_segment_list( Old_area, Seg_list ),
    set_coverage( Sites, Seg_list, New_seg_list ),
    make_polygon_list( New_seg_list, New_area ),
    total_area( Old_area, PA ),
    total_area( New_area, NA ),
    A is PA - NA, !.

network_coverage( [], Area, Area ):- !.
network_coverage( [Station|Station_list], Area, Remaining_area ):-
    network_coverage( Station_list, Area, New_area ),
    set_coverage( [Station|Station_list], New_area, Remaining_area ), !.

set_coverage( [], Area, Area ):- !.
set_coverage( [Station_1, Station_2|List], Area, Remaining_area ):-
    set_coverage( [Station_1|List], Area, New_area ),
    pair_coverage( Station_1, Station_2, New_area, Remaining_area ), !.

pair_coverage( S1, S2, Area, Remaining_area ):-
    coverage_parameters( S1, S2, R1, R2, M ),
    check_range( M, Range ),
    case( [ Range=short -> simple_coverage( S1, S2, R1, R2, M, Coverage ),
           Range=long -> complex_coverage( S1, S2, R1, R2, M, Coverage )
         ], no_coverage( Coverage ) ),
    clip_intersection( Area, Coverage, Remaining_area ), !.

check_range( Scale, too_near ):-
    range_limits( Min, _ ),
    Scale < Min, !.
check_range( Scale, too_far ):-

```

```

    range_limits( _, Max ),
    Scale > Max, !.
check_range( Scale, short ):-
    range_limits( _, Max ),
    Scale < 0.5*Max, !.
check_range( _, long ).

no_coverage( [] ).

simple_coverage( Site_1, Site_2, Rotation_1, Rotation_2, Scale, Coverage ):-
    check_coverage( Site_1, Site_2, Side ),
    coverage( Side, Site_1, Site_2, Rotation_1, Rotation_2, Scale, List ),
    make_segment_list( List, Coverage ), !.

complex_coverage( Site_1, Site_2, Rotation_1, Rotation_2, Scale, Coverage ):-
    simple_coverage( Site_1, Site_2, Rotation_1, Rotation_2, Scale, Max ),
    adjust_for_range( Site_1, Site_2, Max, Coverage ), !.

check_coverage( S1, S2, Side ):-
    linear_parameters( [S1,S2], Baseline ),
    survey_limits( SL ),
    compress_list_of_lists( SL, Point_list ),
    check_baseline( Baseline, Point_list, Side ), !.
check_coverage( _, _, both ).

check_baseline( [A,B,C], [[X,Y]], left ):-
    approx_less_than( (A*X + B*Y + C), 0.0 ), !.
check_baseline( [A,B,C], [[X,Y]], right ):-
    approx_greater_than( (A*X + B*Y + C), 0.0 ), !.
check_baseline( Line, [Point|List], Side ):-
    check_baseline( Line, List, Side ), !,
    check_baseline( Line, [Point], Side ).

coverage( left, Site_1, Site_2, Rotation, _, Scale, [Coverage] ):-
    coverage_prototype( Template ),
    transform_points( Template, Site_1, Rotation, Scale, Coverage ).
coverage( right, Site_1, Site_2, _, Rotation, Scale, [Coverage] ):-
    coverage_prototype( Template ),
    transform_points( Template, Site_2, Rotation, Scale, Coverage ).
coverage( both, S1, S2, R1, R2, S, [Coverage_1,Coverage_2] ):-
    coverage( left, S1, S2, R1, R2, S, [Coverage_1] ),
    coverage( right, S1, S2, R1, R2, S, [Coverage_2] ).

coverage_parameters( Point_1, Point_2, Fwd_az, Back_az, Distance ):-
    polar_parameters( Point_1, Point_2, Fwd_az, Distance ),
    angle_sum( Fwd_az, pi, Back_az ), !.

adjust_for_range( Site_1, Site_2, Simple_coverage, Coverage ):-
    range_polygon( Site_1, Polygon_1 ),
    range_polygon( Site_2, Polygon_2 ),
    polygon_intersection( Polygon_1, Polygon_2, Range_limit ),
    polygon_intersection( Simple_coverage, Range_limit, Coverage ), !.

range_polygon( Site, Polygon ):-
    range_limits( _, Max ),
    range_prototype( Template ),
    transform_points( Template, Site, 0.0, Max, Point_list ),
    make_segment_list( [Point_list], Polygon ), !.

coverage_prototype( [[-0.19,0.14],[-0.5,0.88],[0.0,1.75],
    [1.0,1.75],[1.5,0.88],[1.19,0.14],[-0.19,0.14]] ).

range_prototype( [[1.0,0.0],[0.71,-0.71],[0.0,-1.0],[-0.71,-0.71],

```

```
[-1.0,0.0],[-0.71,0.71],[0.0,1.0],[0.71,0.71],[1.0,0.0]]).
```

```
/*-----  
Cost Function  
-----*/
```

```
cost( [Site_list,_,_], Cost ):-  
    sum_sites( Site_list, Sum ),  
    geometry_function( Site_list, F ),  
    Cost is Sum + F, !.  
  
sum_sites( [], 0.0 ):- !.  
sum_sites( [Site|List], Sum ):-  
    sum_sites( List, S ),  
    station_cost( Site, C ),  
    Sum is S + C, !.  
  
geometry_function( [], 0.0 ):- !.  
geometry_function( [Site], 0.0 ):- !.  
geometry_function( Site_list, F ):-  
    relative_distances( Site_list, Min, Max ),  
    F is 1.0 - Min/Max, !.  
  
relative_distances( [P1,P2], D, D ):-  
    !, magnitude( P1, P2, D ).  
relative_distances( [P|List], Min, Max ):-  
    relative_distances2( [P|List], Min1, Max1 ),  
    relative_distances( List, Min2, Max2 ),  
    ifthenelse( Min1 < Min2,  
        Min = Min1,  
        Min = Min2 ),  
    ifthenelse( Max1 > Max2,  
        Max = Max1,  
        Max = Max2 ).  
  
relative_distances2( [P1,P2], D, D ):-  
    !, magnitude( P1, P2, D ).  
relative_distances2( [P1,P2|List], Min, Max ):-  
    magnitude( P1, P2, D ),  
    relative_distances2( [P1|List], Min1, Max1 ),  
    ifthenelse( Min1 < D,  
        Min = Min1,  
        Min = D ),  
    ifthenelse( Max1 > D,  
        Max = Max1,  
        Max = D ).
```

/*=====

EVALUATION FUNCTION DEFINITIONS

Original code by Arnold Steed, May 1991

Note: Only one of the following functions should be
in the database at a time.

=====*/

/*-----
Heuristic Evaluation Function version 1
-----*/

```
eval( [[_,_,_], E ):-  
    !, estimated_site_cost( Cs ),  
    E is 2*Cs.  
eval( [[_],_,_], E ):-  
    !, estimated_site_cost( E ).  
eval( [_C,RA], E ):-  
    total_area( RA, A ),  
    N1 is A/C,  
    length( RA, N2 ),  
    evaluation_weights( W1, W2 ),  
    estimated_site_cost( S ),  
    ifthenelse( N1 = err,  
        E is 100.0,  
        E is S*(W1*N1 + W2*N2) ), !.
```

/*-----
Heuristic Evaluation Function version 2
-----*/

```
eval( [[_,_,_], E ):-  
    !, estimated_site_cost( Cs ),  
    E is 2*Cs.  
eval( [[_],_,_], E ):-  
    !, estimated_site_cost( E ).  
eval( [_C,RA], E ):-  
    total_area( RA, A ),  
    N1 is A/C,  
    length( RA, N2 ),  
    estimated_site_cost( S ),  
    ifthenelse( N1 = err,  
        E is 100.0,  
        (max( [N1,N2], N ), E is S*N) ), !.
```

/*-----
A* Evaluation Function
-----*/

```
eval( [_,_], 0 ):- !.  
eval( _, 1 ).
```

```
/*=====
```

CENTER OF POLYGON FUNCTION DEFINITIONS

Original code by Arnold Steed, May 1991

Note: Only one of the following functions should be
in the database at a time.

```
=====*/
```

```
/*-----
```

Center of Polygonversion 1

```
-----*/
```

```
center_of_polygon( Polygon, Center ):-
    convex_center( Polygon, Center ).
    make_segment_list( [Polygon], Segment_list ),
    point_in_polygon( Center, Segment_list ), !.
center_of_polygon( Polygon, Center ):-
    concave_center( Polygon, Center ).
center_of_polygon( Polygon, _ ):-
    writeln( 'version 1 of center_of_polygon failed on polygon' ),
    signal( Polygon ),
    fail.
```

```
convex_center( [_|Polygon], [X,Y] ):-
    sum_points( Polygon, X_sum, Y_sum ),
    length( Polygon, N ),
    X is integer( X_sum/N + 0.5 ),
    Y is integer( Y_sum/N + 0.5 ), !.
```

```
concave_center( Polygon, Center ):-
    largest_triangle( Polygon, [A,B,C] ), !,
    convex_center( [A,B,C,A], Center ), !.
```

```
largest_triangle( [A,B,C,A], [A,B,C] ):- !.
largest_triangle( [A,B,C|Polygon], Triangle ):-
    largest_triangle( [A,C|Polygon], New_triangle ),
    special_triangle_area( New_triangle, New_area ),
    special_triangle_area( [A,B,C], Old_area ),
    ifthenelse( New_area > Old_area,
        Triangle = New_triangle,
        Triangle = [A,B,C] ), !.
```

```
sum_points( [], 0, 0 ):- !.
sum_points( [[X,Y]|Points], X_sum, Y_sum ):-
    sum_points( Points, X_add, Y_add ),
    X_sum is X + X_add,
    Y_sum is Y + Y_add.
```

```
/*-----
```

Center of Polygonversion 2

```
-----*/
```

```
center_of_polygon( Polygon, [X,Y] ):-
    weighted_sum( Polygon, Area, X_sum, Y_sum ),
    X is integer( X_sum/Area + 0.5 ),
    Y is integer( Y_sum/Area + 0.5 ), !.
center_of_polygon( Polygon, _ ):-
    writeln( 'version 2 of center_of_polygon failed on polygon' ),
    signal( Polygon ),
    fail.
```

```

weighted_sum( [A,B,A], 0.0, 0, 0 ):- !.
weighted_sum( [A,B,C|Points], Area, X_sum, Y_sum ):-
    weighted_sum( [A,C|Points], Part_area, Part_X, Part_Y ),
    special_triangle_area( [A,B,C], Tri_area ),
    center_of_triangle( [A,B,C], [Tri_X,Tri_Y] ),
    Area is Part_area + Tri_area,
    X_sum is Part_X + Tri_area*Tri_X,
    Y_sum is Part_Y + Tri_area*Tri_Y, !.

center_of_triangle( [[X1,Y1],[X2,Y2],[X3,Y3]], [X,Y] ):-
    X is integer( (X1 + X2 + X3)/3 + 0.5 ),
    Y is integer( (Y1 + Y2 + Y3)/3 + 0.5 ).

```

```
/*=====
```

GEOMETRY PREDICATES

Original Code by Arnold Steed, June 1991

Additional Code by Neil Rowe

```
=====*/
```

```
/*-----
```

Polygon functions requiring segment-list representation

```
-----*/
```

```

polygon_intersection( Polygon_1, Polygon_2, Intersection ):-
    trim_outside( Polygon_2, Polygon_1, Trim_list_1 ),
    trim_outside( Polygon_1, Polygon_2, Trim_list_2 ),
    append( Trim_list_1, Trim_list_2, Intersection ), !.

polygon_union( Polygon_1, Polygon_2, Union ):-
    trim_inside( Polygon_2, Polygon_1, Trim_list_1 ),
    trim_inside( Polygon_1, Polygon_2, Trim_list_2 ),
    append( Trim_list_1, Trim_list_2, Union ), !.

clip_intersection( Polygon_to_clip, Test_polygon, Clipped_polygon ):-
    trim_inside( Test_polygon, Polygon_to_clip, Trim_list_1 ),
    trim_outside( Polygon_to_clip, Test_polygon, Trim_list_2 ),
    reverse_direction( Trim_list_2, Rev_trim_list ),
    append( Trim_list_1, Rev_trim_list, Clipped_polygon ), !.

trim_inside( [], Polygon, Polygon ):- !.
trim_inside( _, [], [] ):- !.
trim_inside( Test_polygon, [Segment|Polygon], Output_polygon ):-
    trim_inside( Test_polygon, Polygon, Partial_output ),
    find_sorted_intersections( Segment, Test_polygon, Point_list ),
    trim_inside_segments( Test_polygon, Point_list, Trim_segments ),
    append( Trim_segments, Partial_output, Output_polygon ), !.

trim_outside( [], _, [] ):- !.
trim_outside( _, [], [] ):- !.
trim_outside( Test_polygon, [Segment|Polygon], Output_polygon ):-
    trim_outside( Test_polygon, Polygon, Partial_output ),
    find_sorted_intersections( Segment, Test_polygon, Point_list ),
    trim_outside_segments( Test_polygon, Point_list, Trim_segments ),
    append( Trim_segments, Partial_output, Output_polygon ), !.

find_sorted_intersections( Segment, Polygon, Point_list ):-
    seg_poly_intersect( Segment, Polygon, List_1 ),
    linear_parameters( Segment, Line ),
    check_endpoints( Polygon, Line, List_1, List_2 ),
    sort_intersections( Segment, List_2, Point_list ), !.

trim_inside_segments( Test_polygon, [P1,P2|Point_list],
    Segment_list ):-
    point_on_polygon( P1, Test_polygon ),
    midpoint( [P1,P2], P ),
    point_in_polygon( P, Test_polygon ), !,
    trim_segments( [P2|Point_list], Segment_list ), !.
trim_inside_segments( Test_polygon, [P|Point_list], Segment_list ):-
    point_on_polygon( P, Test_polygon ), !,
    trim_segments( [P|Point_list], Segment_list ), !.
trim_inside_segments( Test_polygon, [P|Point_list], Segment_list ):-
    point_in_polygon( P, Test_polygon ), !,
    trim_segments( Point_list, Segment_list ), !.

```

```

trim_inside_segments( _, Point_list, Segment_list ):-
    trim_segments( Point_list, Segment_list ).

trim_outside_segments( Test_polygon, [P1,P2|Point_list],
                        Segment_list ):-
    point_on_polygon( P1, Test_polygon ),
    midpoint( [P1,P2], P ),
    point_in_polygon( P, Test_polygon ), !,
    trim_segments( [P1,P2|Point_list], Segment_list ), !.
trim_outside_segments( Test_polygon, [P|Point_list], Segment_list ):-
    point_on_polygon( P, Test_polygon ), !,
    trim_segments( Point_list, Segment_list ), !.
trim_outside_segments( Test_polygon, [P|Point_list], Segment_list ):-
    point_in_polygon( P, Test_polygon ), !,
    trim_segments( [P|Point_list], Segment_list ), !.
trim_outside_segments( _, [P|Point_list], Segment_list ):-
    trim_segments( Point_list, Segment_list ).

trim_segments( [], [] ):- !.
trim_segments( [], [] ):- !.
trim_segments( [P1,P2|Point_list], [[P1,P2]|Segment_list] ):-
    trim_segments( Point_list, Segment_list ).

sort_intersections( Point_list, [], Point_list ):- !.
sort_intersections( [P1,P2], [P3|Points], New_list ):-
    ( same_point( P1, P3 ); same_point( P2, P3 ) ), !,
    sort_intersections( [P1,P2], Points, New_list ), !.
sort_intersections( [P1,P2], [P3|Points], New_list ):-
    find_points_between( [P1,P3], Points, List_1 ),
    find_points_between( [P3,P2], Points, List_2 ),
    sort_intersections( [P1,P3], List_1, Sorted_list_1 ),
    sort_intersections( [P3,P2], List_2, [P3|Sorted_list_2] ),
    append( Sorted_list_1, Sorted_list_2, New_list ), !.

find_points_between( _, [], [] ):- !.
find_points_between( [P1,P2], [P|Points], [P|New_points] ):-
    between_points( P1, P2, P ), !,
    find_points_between( [P1,P2], Points, New_points ), !.
find_points_between( End_points, [_|Points], New_points ):-
    find_points_between( End_points, Points, New_points ).

reverse_direction( [], [] ):- !.
reverse_direction( [[P1,P2]|In_list], [[P2,P1]|Out_list] ):-
    reverse_direction( In_list, Out_list ).

/*-----
    Line, Segment, and Ray Intersections
    (polygons represented as segment lists)
-----*/

line_intersection( [A1,B1,C1], [A2,B2,C2], _ ):-
    approx_equal_to( B1*A2, B2*A1 ), !,
    approx_equal_to( C2*A1, C1*A2 ),
    approx_equal_to( C2*B1, C1*B2 ),
    writeln( 'Warning: Colinear lines may cause errors in polygons.' ),
    write_list( ['Lines:', [A1,B1,C1], [A2,B2,C2]] ),
    fail.
line_intersection( [A1,B1,C1], [A2,B2,C2], [X,Y] ):-
    D is B1*A2 - B2*A1,
    X is (C1*B2 - C2*B1)/D,
    Y is (C2*A1 - C1*A2)/D,
    ifthen( (X=err;Y=err),
        (writeln('Line intersection error.')),

```

```

        writeln( [A1,B1,C1] ),
        writeln( [A2,B2,C2] ),
        fail) ), !.

line_circle_intersect( [A,B,C], [[X0,Y0],R], [X,Y] ):-
    approx_equal_to( B, 0.0 ), !,
    X is -C/A,
    Qa is 1.0,
    Qb is -2.0*Y0,
    Qc is (X - X0)^2 + Y0^2 - R^2,
    solve_quadratic( Qa, Qb, Qc, Y ).
line_circle_intersect( [A,B,C], [[X0,Y0],R], [X,Y] ):-
    approx_equal_to( A, 0.0 ), !,
    Y is -C/B,
    Qa is 1.0,
    Qb is -2.0*X0,
    Qc is (Y - Y0)^2 + X0^2 - R^2,
    solve_quadratic( Qa, Qb, Qc, X ).
line_circle_intersect( [A,B,C], [[X0,Y0],R], [X,Y] ):-
    AB is A/B,
    Bf is Y0 + C/B,
    Qa is 1 + AB^2,
    Qb is 2*(AB*Bf - X0),
    Qc is Bf^2 + X0^2 - R^2,
    solve_quadratic( Qa, Qb, Qc, X ),
    Y is -(A*X + C)/B.

seg_circle_intersect( [P1,P2], Circle, [X,Y] ):-
    linear_parameters( [P1,P2], Line ), !,
    line_circle_intersect( Line, Circle, [FX,FY] ),
    between_points( P1, P2, [FX,FY] ),
    X is integer( FX + 0.5 ),
    Y is integer( FY + 0.5 ).

segment_intersection( [P1,P2], [P3,P4], [X,Y] ):-
    linear_parameters( [P1,P2], Line_1 ),
    linear_parameters( [P3,P4], Line_2 ),
    line_intersection( Line_1, Line_2, [FX,FY] ),
    between_points( P1, P2, [FX,FY] ),
    between_points( P3, P4, [FX,FY] ),
    X is integer( FX + 0.5 ),
    Y is integer( FY + 0.5 ), !.

ray_line_intersect( Ray, Line, Intersect ):-
    ray_to_line( Ray, Rline ),
    line_intersection( Line, Rline, Intersect ),
    point_on_ray( Intersect, Ray ), !.

ray_seg_intersect( Ray, [P1,P2], [X,Y] ):-
    linear_parameters( [P1,P2], Line ),
    ray_line_intersect( Ray, Line, [FX,FY] ),
    between_points( P1, P2, [FX,FY] ),
    X is integer( FX + 0.5 ),
    Y is integer( FY + 0.5 ), !.

seg_poly_intersect( _, [], [] ):- !.
seg_poly_intersect( Segment_1, [Segment_2|Polygon], [P|Points] ):-
    segment_intersection( Segment_1, Segment_2, P ), !,
    seg_poly_intersect( Segment_1, Polygon, Points ).
seg_poly_intersect( Segment, [_|Polygon], Points ):-
    seg_poly_intersect( Segment, Polygon, Points ).

ray_poly_intersect( _, [], [] ):- !.

```

```

ray_poly_intersect( Ray, [Segment|Polygon], [P|Points] ):-
    ray_seg_intersect( Ray, Segment, P ), !,
    ray_poly_intersect( Ray, Polygon, Points ).
ray_poly_intersect( Ray, [_|Polygon], Points ):-
    ray_poly_intersect( Ray, Polygon, Points ).

/*-----
    Point Location Functions
    (polygons are represented as segment lists)
-----*/

point_on_polygon( Point, [Segment|Polygon] ):-
    point_on_segment( Point, Segment ), !.
point_on_polygon( Point, [_|Polygon] ):-
    point_on_polygon( Point, Polygon ).

point_on_segment( Point, [P1,P2] ):-
    linear_parameters( [P1,P2], Line ),
    point_on_line( Point, Line ),
    between_points( P1, P2, Point ), !.

point_on_ray( P, [P0,Dir] ):-
    direction( P0, P, PDir ),
    approx_equal_to( Dir, PDir ), !.

point_on_line( [X,Y], [A,B,C] ):-
    approx_equal_to( A*X+B*Y+C, 0.0 ).

point_in_polygon( [X,Y], Polygon ):-
    ray_poly_intersect( [[X,Y],0.0], Polygon, Point_list ),
    ray_to_line( [[X,Y],0.0], Line ),
    check_endpoints( Polygon, Line, Point_list, New_list ),
    length( New_list, N ),
    odd( N ), !.

between_points( [X1,Y1], [X2,Y2], [X,Y] ):-
    between( X1, X2, X ),
    between( Y1, Y2, Y ), !.

between( N1, N2, N ):-
    approx_less_than( N1, N ),
    approx_less_than( N, N2 ), !.
between( N1, N2, N ):-
    approx_less_than( N, N1 ),
    approx_less_than( N2, N ), !.

same_point( [X,Y], [X,Y] ):-
    integer( X ),
    integer( Y ).

midpoint( [[X1,Y1],[X2,Y2]], [X,Y] ):-
    X is integer( (X1 + X2)/2 + 0.5 ),
    Y is integer( (Y1 + Y2)/2 + 0.5 ).

nearest_point_on_line( [A,B,C], [X0,Y0], [X,Y] ):-
    C0 is A*Y0 - B*X0,
    line_intersection( [A,B,C], [B,-A,C0], [X,Y] ), !.

nearest_point_on_segment( [P1,P2], P0, [X,Y] ):-
    linear_parameters( [P1,P2], Line ),
    nearest_point_on_line( Line, P0, [FX,FY] ),
    between_points( P1, P2, [FX,FY] ), !,
    X is integer( FX + 0.5 ),

```

```

Y is integer( FY + 0.5 ).
nearest_point_on_segment( [P1,P2], P0, P ):-
    magnitude( P1, P0, M1 ),
    magnitude( P2, P0, M2 ),
    ifthenelse( M1 < M2, P = P1, P = P2 ), !.

nearest_point_on_list( [Segment], P0, P ):-
    !, nearest_point_on_segment( Segment, P0, P ).
nearest_point_on_list( [Segment|List], P0, P ):-
    nearest_point_on_segment( Segment, P0, P1 ),
    nearest_point_on_list( List, P0, P2 ),
    magnitude( P1, P0, M1 ),
    magnitude( P2, P0, M2 ),
    ifthenelse( M1 < M2, P = P1, P = P2 ), !.

delete_point( P1, [P1,P2|Points], Polygon ):-
    !, del one instance( P1, Points, Temp ),
    append( [P2|Temp], [P2], Polygon ).
delete_point( P, Old_polygon, New_polygon ):-
    del_one_instance( P, Old_polygon, New_polygon ).

/*-----
                        Point Transformations
-----*/

transform_points( [], [], [] ):- !.
transform_points([[[X,Y]|List], [X0,Y0], Rotation, Scale,
    [[XT,YT]|T_list] ):-
    transform_points( List, [X0,Y0], Rotation, Scale, T_list ),
    polar_parameters( [0,0], [X,Y], Direction, Length ),
    angle_sum( Direction, Rotation, Angle ),
    Magnitude is Length*Scale,
    XT is integer( Magnitude*cos( Angle ) + X0 + 0.5 ),
    YT is integer( Magnitude*sin( Angle ) + Y0 + 0.5 ), !.

/*-----
                        Structure Type Conversions
-----*/

make_segment_list( [], [] ):- !.
make_segment_list( [P|Polygons], Segment_list ):-
    make_segment_list( Polygons, Segments ),
    make_segments( P, S ),
    append( S, Segments, Segment_list ), !.

make_segments( [], [] ):- !.
make_segments( [P1,P2|Points], [[P1,P2]|Segments] ):-
    make_segments( [P2|Points], Segments ).

make_point_list( [[P1,P2]], [P1,P2] ):- !.
make_point_list( [[P1,P2]|Segments], [P1,P2|Points] ):-
    find_next_segment( P2, Segments, New_seg_list ),
    make_point_list( New_seg_list, [P2|Points] ), !.

find_next_segment( P1, [[P1,P2]|Segments], [[P1,P2]|Segments] ):-!.
find_next_segment( P1, [Segment|Segments], New_seg_list ):-
    append( Segments, [Segment], Seg_list ),
    find_next_segment( P1, Seg_list, New_seg_list ), !.

make_polygon_list( Segment_list, Polygon_list ):-
    separate_polygons( Segment_list, Polygon_segments ),
    make_polygons( Polygon_segments, Polygon_list ), !.

```

```

separate_polygons( [], [] ):- !.
separate_polygons( [[P1,P2]|Segments],
    [[P1,P2]|Polygon]|Polygons] ):-
    separate_one_polygon( P2, Segments, Polygon, New_list ),
    separate_polygons( New_list, Polygons ), !.

separate_one_polygon( P1, Segments, [[P1,P2]|Polygon], Remains ):-
    del_one_instance( [P1,P2], Segments, New_list ), !,
    separate_one_polygon( P2, New_list, Polygon, Remains ).
separate_one_polygon( _, Segments, [], Segments ).

make_polygons( [], [] ):- !.
make_polygons( [Segments|Seg_list], [Polygon|Poly_list] ):-
    make_point_list( Segments, Polygon ),
    make_polygons( Seg_list, Poly_list ), !.

ray_to_line( [[X,Y],D], [A,B,C] ):-
    A is sin(D),
    B is -cos(D),
    C is -(A*X + B*Y), !.

/*-----
    Check for Intersections at Vertices
-----*/

check_endpoints( Polygon, Line, Point_list, New_point_list ):-
    multiple_occurrence( Point, Point_list ), !,
    member( [Point_1,Point], Polygon ),
    member( [Point_1,Point_2], Polygon ),
    check_intersections( Point_1, Point_2, Line, Flag ),
    ifthenelse( (Flag > 0),
        (del_all_instances( Point, Point_list, New_list )),
        (del_one_instance( Point, Point_list, New_list )) ),
    !, check_endpoints( Polygon, Segment, New_list, New_point_list ).
check_endpoints( _, _, Point_list, Point_list ).

check_intersections( [X1,Y1], [X2,Y2], [A,B,C], Flag ):-
    F1 is A*X1 + B*Y1 + C,
    F2 is A*X2 + B*Y2 + C,
    Flag is F1*F2, !.

/*-----
    Area of Polygons
    (polygons represented as point lists)
-----*/

total_area( [], 0 ):- !.
total_area( [Polygon|Polygons], Total_area ):-
    total_area( Polygons, Area_2 ),
    polygon_area( Polygon, Area_1 ),
    Total_area is Area_1 + Area_2, !.

polygon_area( [A,B,A], 0.0 ):- !.
polygon_area( [A,B,C|Points], Area ):-
    polygon_area( [A,C|Points], Part_area ),
    special_triangle_area( [A, B, C], Tri_area ),
    Area is Part_area + Tri_area, !.

special_triangle_area( [A,B,C], Area ):-
    oblique_angle( A, B, C ), !,
    triangle_area( [A,B,C], Tri_area ),
    Area is Tri_area, !.
special_triangle_area( [A,B,C], Area ):-

```

```

triangle_area( [A,B,C], Area ), !.

triangle_area( [P1,P2,P3], Area ):-
    magnitude( P1, P2, A ),
    magnitude( P2, P3, B ),
    magnitude( P3, P1, C ),
    S is (A+B+C)/2,
    S2 is S*(S-A)*(S-B)*(S-C),
    Area is sqrt( S2 ), !.

/*-----
                        Angle Functions

Original Code by Neil Rowe
Modified by Arnold Steed, June 1991
-----*/

angle_of_intersection( [X1,Y1], [X2,Y2], [X,Y], Angle ):-
    direction( [X,Y], [X1,Y1], D1 ),
    direction( [X,Y], [X2,Y2], D2 ),
    A1 is D2 - D1,
    smaller_angle( A1, Angle ), !.

smaller_angle( Angle, Small_angle ):-
    A is abs( Angle ),
    ifthenelse( A > pi,
        Small_angle is 2*pi - A,
        Small_angle is A ), !.

angle_difference( A1, A2, DA ):-
    DA12 is A1 - A2,
    normalize_angle( DA12, DA ).

angle_sum( A1, A2, DA ):-
    DA12 is A1 + A2,
    normalize_angle( DA12, DA ).

before( D1, D2 ):-
    angle_difference( D2, D1, D ),
    T is sin( D ),
    T >= 0.0, !.

normalize_angle( A, NA ):-
    Twopi is 2*pi,
    floatmod( A, Twopi, NA ), !.

find_oblique_angle( PL, P1, P2, P3 ):-
    circular_sequence( PL, P1, P2 ),
    circular_sequence( PL, P2, P3 ),
    oblique_angle( P1, P2, P3 ), !.

find_non_oblique_angle( PL, P1, P2, P3 ):-
    circular_sequence( PL, P1, P2 ),
    circular_sequence( PL, P2, P3 ),
    non_oblique_angle( P1, P2, P3 ), !.

circular_sequence( [P1,P2|Point_list], P1, P2 ).
circular_sequence( [P|Point_list], P1, P2 ):-
    circular_sequence( Point_list, P1, P2 ).

oblique_angle( P1, P2, P3 ):-
    direction( P1, P2, H12 ),
    direction( P2, P3, H23 ),

```

```

before( H12, H23 ), !.

non_oblique_angle( P1, P2, P3 ):-
    direction( P1, P2, H12 ),
    direction( P2, P3, H23 ),
    before( H23, H12 ), !.

/*-----
    Center of largest polygon in list
    (requires point-list representation)
-----*/

center_of_area( Polygon_list, Center ):-
    largest_polygon( Polygon_list, Polygon ),
    center_of_polygon( Polygon, Center ), !.

largest_polygon( [Polygon], Polygon ):-!.
largest_polygon( [Poly1|List], Polygon ):-
    largest_polygon( List, Poly2 ),
    polygon_area( Poly1, Area1 ),
    polygon_area( Poly2, Area2 ),
    ifthenelse( Area1 > Area2,
        ( Polygon = Poly1 ),
        ( Polygon = Poly2 ) ), !.

/*-----
    Miscellaneous utility predicates
-----*/

most_distant_vertex( Polygon_list, Center, Vertex ):-
    farthest_in_poly_list( Polygon_list, Center, Vertex, _ ).

farthest_in_poly_list( [Polygon], C, P, M ):-
    !, farthest_in_point_list( Polygon, C, P, M ).
farthest_in_poly_list( [Polygon|L], C, P, M ):-
    farthest_in_poly_list( L, C, P1, M1 ),
    farthest_in_point_list( Polygon, C, P2, M2 ),
    ifthenelse( M1 > M2,
        ( P = P1, M = M1 ),
        ( P = P2, M = M2 ) ), !.

farthest_in_point_list( [P], C, P, M ):-
    magnitude( P, C, M ), !.
farthest_in_point_list( [P1|Points], C, P, M ):-
    farthest_in_point_list( Points, C, P2, M2 ),
    magnitude( P1, C, M1 ),
    ifthenelse( M1 > M2,
        ( P = P1, M = M1 ),
        ( P = P2, M = M2 ) ), !.

get_radial_intersections( P, I ):-
    radial_intersections( P, 360, I ).

radial_intersections( P, 0, [] ):- !.
radial_intersections( P, Dir, I_list ):-
    Dir2 is Dir - 20,
    radial_intersections( P, Dir2, Part_list ),
    shore_segments(S),
    Rad is pi*Dir/180,
    ray_poly_intersect( [P,Rad], S, Points ),
    closest_point( P, Points, I ),
    append( I, Part_list, I_list ), !.

```

```

closest_point( _, [], [] ):-!.
closest_point( _, [P], [P] ):- !.
closest_point( P0, [P1|Points], [P] ):-
    closest_point( P0, Points, [P2] ),
    magnitude( P0, P1, M1 ),
    magnitude( P0, P2, M2 ),
    ifthenelse( M1 < M2, P = P1, P = P2 ), !.

```

```
/*=====
```

MATH PREDICATES

Original Code by Arnold Steed, May 1991

Additional Code by Neil Rowe

```
=====*/
```

```
min( [X,Y], X ):-
    X < Y, !.
min( [X,Y], Y ):- !.
min( [X|List], Min ):-
    min( List, M1 ),
    min( [X,M1], Min ).

max( [X,Y], X ):-
    X > Y, !.
max( [X,Y], Y ):- !.
max( [X|List], Min ):-
    max( List, M1 ),
    max( [X,M1], Min ).

approx_less_than( X, Y ):-
    X - Y < (abs(X)+abs(Y))*1.0e-14, !.
approx_less_than( X, Y ):-
    X - Y < 1.0e-14.

approx_greater_than( X, Y ):-
    Y - X < (abs(X)+abs(Y))*1.0e-14, !.
approx_greater_than( X, Y ):-
    Y - X < 1.0e-14.

approx_equal_to( X, Y ):-
    abs( X - Y ) < (abs(X)+abs(Y))*1.0e-14, !.
approx_equal_to( X, Y ):-
    abs( X - Y ) < 1.0e-14.

floatmod( X, M, X ):-
    X < M, X >= 0, !.
floatmod( X, M, Y ):-
    X < 0, !,
    MX is 0 - X,
    floatmod( MX, M, MY ),
    Y is M - MY.
floatmod( X, M, Y ):-
    NX is X - M,
    floatmod( NX, M, Y ), !.

odd( N ):-
    integer( N ),
    (N/2) =\= (N//2).

slope_intercept( [[X1,Y1],[X2,Y2]], [Slope,Intercept] ):-
    X1 =\= X2,
    Slope is (Y2 - Y1)/(X2 - X1),
    Intercept is Y1 - Slope*X1.

linear_parameters( [[X,Y],[X,Y]], _ ):-
    !, write( 'Warning: Degenerate line at ' ),
    write( [X,Y] ),
    writeln( ' cannot be parametrized.' ),
    fail.
```

```

linear_parameters( [[X1,Y1],[X2,Y2]], [A,B,C] ):-
    An is Y2 - Y1,
    Bn is X1 - X2,
    D is sqrt( An*An + Bn*Bn ),
    A is An/D,
    B is Bn/D,
    C is -(A*X2 + B*Y2).

polar_parameters( [X,Y], [X,Y], 0.0, 0.0 ):- !.
polar_parameters( [X1,Y1], [X2,Y2], Angle, Magnitude ):-
    direction( [X1,Y1], [X2,Y2], Angle ),
    magnitude( [X1,Y1], [X2,Y2], Magnitude ).

direction( [X,Y], [X,Y], _ ):-
    write_list( ['Degenerate line at',[X,Y],'has no direction.'] ), nl,
    !, fail.
direction( [X1,Y1], [X2,Y2], D ):-
    Y2 < Y1, X2 < X1,
    direction2( [X1,Y1], [X2,Y2], D2 ),
    D is D2 + pi, !.
direction( [X1,Y1], [X2,Y2], D ):-
    Y2 < Y1,
    direction2( [X1,Y1], [X2,Y2], D2 ),
    D is (2*pi) - D2, !.
direction( [X1,Y1], [X2,Y2], D ):-
    X2 < X1,
    direction2( [X1,Y1], [X2,Y2], D2 ),
    D is pi - D2, !.
direction( [X1,Y1], [X2,Y2], D ):-
    direction2( [X1,Y1], [X2,Y2], D ), !.

direction2( [X1,Y1], [X2,Y2], D ):-
    DX is X1 - X2, DY is Y1 - Y2,
    ADX is abs(DX), ADY is abs(DY),
    ADX > ADY, Q is ADY/ADX,
    D is atan(Q), !.
direction2( [X1,Y1], [X2,Y2], D ):-
    DX is X1 - X2, DY is Y1 - Y2,
    ADX is abs(DX), ADY is abs(DY),
    Q is ADX / ADY,
    D is (pi/2) - atan(Q), !.

magnitude( [X1,Y1], [X2,Y2], Magnitude ):-
    DX is X2 - X1, DY is Y2 - Y1,
    M2 is DX^2 + DY^2,
    Magnitude is sqrt( M2 ).

random_integer( M, N, R ):-
    D is N - M + 1,
    R is integer( random*D + M ).

mean_square_error( numbers, List_1, List_2, Drms ):-
    vector_subtract( List_1, List_2, Residuals ),
    sum_squares( Residuals, Sum ),
    length( List_1, N ),
    Drms is sqrt( Sum/N ).

mean_square_error( points, List_1, List_2, Drms ):-
    position_differences( List_1, List_2, Residuals ),
    sum_squares( Residuals, Sum ),
    length( List_1, N ),
    Drms is sqrt( Sum/N ).

sum_squares( [], 0 ):- !.

```

```

sum_squares( [X|X_list], Sum ):-
    Z is X^2,
    sum_squares( X_list, A_sum ),
    Sum is A_sum + Z.

position_differences( [], [], [] ):- !.
position_differences( [P1|List_1], [P2|List_2], [M|M_list] ):-
    position_differences( List_1, List_2, M_list ),
    magnitude( P1, P2, M ).

dot_product( [], [], 0 ):- !.
dot_product( [X|X_list], [Y|Y_list], Z ):-
    dot_product( X_list, Y_list, Zp ),
    Z is Zp + X*Y.

scalar_multiply( _, [], [] ):- !.
scalar_multiply( A, [X|X_list], [Y|Y_list] ):-
    scalar_multiply( A, X_list, Y_list ),
    Y is A*X.

vector_add( [], [], [] ):- !.
vector_add( [X|X_list], [Y|Y_list], [Z|Z_list] ):-
    vector_add( X_list, Y_list, Z_list ),
    Z is X + Y.

vector_subtract( [], [], [] ):- !.
vector_subtract( [X|X_list], [Y|Y_list], [Z|Z_list] ):-
    vector_subtract( X_list, Y_list, Z_list ),
    Z is X - Y.

solve_quadratic( A, B, C, X ):-
    D is B*B - 4*A*C,
    solve_quadratic( A, B, C, D, X ).
solve_quadratic( _, _, _, D, _ ):-
    D < 0, !,
    fail.
solve_quadratic( A, B, C, D, X ):-
    approx_equal_to( D, 0.0 ), !,
    X is -B/(2*A).
solve_quadratic( A, B, C, D, X ):-
    X is -(B - sqrt(D))/(2*A).
solve_quadratic( A, B, C, D, X ):-
    X is -(B + sqrt(D))/(2*A).

```

/*=====

BASIC PREDICATES

Original code by Arnold Steed, May 1991
Additional code by Neil Rowe and Robert Marks

=====*/

```
writeln( X ):-
    write( X ), nl.

writeln( S, X ):-
    write( S, X ), nl( S ).

write_list( [] ):-
    nl.
write_list( [Element|List] ):-
    write( Element ),
    write( ' ' ),
    write_list( List ).

write_list( S, [] ):-
    nl( S ).
write_list( S, [Element|List] ):-
    write( S, Element ),
    write( S, ' ' ),
    write_list( S, List ).

same( X, X ).

compress_list_of_lists( [], [] ):- !.
compress_list_of_lists( [List], List ):- !.
compress_list_of_lists( [List|Lists], New_list ):-
    compress_list_of_lists( Lists, Partial_list ),
    append( List, Partial_list, New_list ).

list_element( [Head|Tail], Head, 1 ).
list_element( [_Head|Tail], Element, N ):-
    var( N ),
    list_element( Tail, Element, Np ),
    N is Np + 1.
list_element( [_Head|Tail], Element, N ):-
    var( Element ),
    Np is N - 1,
    list_element( Tail, Element, Np ).

member( Element, [Element|_List] ).
member( Element, [_Head|Tail] ):-
    member( Element, Tail ).

multiple_occurance( Element, [Element|List] ):-
    member( Element, List ).
multiple_occurance( Element, [_Head|Tail] ):-
    multiple_occurance( Element, Tail ).

del_one_instance( Element, [Element|List], List ).
del_one_instance( Element, [Head|List_1], [Head|List_2] ):-
    del_one_instance( Element, List_1, List_2 ).

del_all_instances( _, [], [] ).
del_all_instances( Element, [Element|Tail], List ):-
    del_all_instances( Element, Tail, List ).
```

```

del_all_instances( Element, [Head|Tail_1], [Head|Tail_2] ):-
    del_all_instances( Element, Tail_1, Tail_2 ).

append( [], List, List ).
append( [Head|Tail], List, [Head|New_tail] ):-
    append( Tail, List, New_tail ).

write_current_time( Time ):-
    time( Time ),
    write_time( Time ).
write_current_time( H, Time ):-
    time( Time ),
    write_time( H, Time ).

write_time( time(H,M,S,HS) ):-
    Sec is S + HS/100,
    write( H ), write( ':' ),
    write( M ), write( ':' ),
    write( Sec ).
write_time( F, time(H,M,S,HS) ):-
    Sec is S + HS/100,
    write( F, H ), write( F, ':' ),
    write( F, M ), write( F, ':' ),
    write( F, Sec ).

write_current_date( Date ):-
    date( Date ),
    write_date( Date ), !.
write_current_date( H, Date ):-
    date( Date ),
    write_date( H, Date ), !.

write_date( date(Y,M,D) ):-
    write( M ), write( '-' ),
    write( D ), write( '-' ),
    write( Y ), write( ' ' ), !.
write_date( H, date(Y,M,D) ):-
    write( H, M ), write( H, '-' ),
    write( H, D ), write( H, '-' ),
    write( H, Y ), write( H, ' ' ), !.

elapsed_time( time(H1,M1,S1,HS1), time(H2,M2,S2,HS2), T ):-
    H1 < H2, !,
    Hp is H1 + 24,
    elapsed_time( time(Hp,M1,S1,HS1), time(H2,M2,S2,HS2), T ).
elapsed_time( T1, T2, T ):-
    convert_time( T1, S1 ),
    convert_time( T2, S2 ),
    S is S1 - S2,
    convert_time( T, S ).

convert_time( time(H,M,S,HS), T ):-
    var(T), !,
    T is H*3600 + M*60 + S + HS/100.
convert_time( time(H,M,S,HS), T ):-
    H is integer(T/3600),
    M is integer((T - H*3600)/60),
    S is integer(T - (H*3600 + M*60)),
    HS is integer(100*(T - (H*3600 + M*60 + S))).

set_global( Name, Value ):-
    abolish( Name/1 ),
    P=..[Name,Value],

```

```

    asserta( P ), !.
get_global( Name, Value ):-
    P=..[Name,Value],
    call( P ), !.
cons_global( Name, I ):-
    P=..[Name,X],
    call( P ),
    retract( P ),
    NP=..[Name,[I|X]],
    asserta( NP ), !.
pop_global( Name, Value ):-
    P=..[Name,[Value|L]],
    call( P ),
    retract( P ),
    NP=..[Name,L],
    asserta( NP ), !.
inc_global( Name ):-
    P=..[Name,X],
    call( P ),
    retract( P ),
    Xp1 is X + 1,
    NP=..[Name,Xp1],
    asserta( NP ), !.

beep:-
    put(7).

beep( N ):-
    ctr_set(0,1),
    repeat,
    ctr_inc(0,I),
    [! beep, delay( 0.25 ) !],
    I =:= N.

delay( Seconds ):-
    time( time( H, M, S, D ) ),
    Time1 is float( H*3600 + M*60 + S + D/100.0 ),
    repeat,
    time( time( H2, M2, S2, D2 ) ),
    Time2 is float( H2*3600 + M2*60 + S2 + D2/100.0 ),
    Time1 + Seconds =< Time2.

signal( Message ):-
    writeln( Message ), nl,
    writeln( '< Press any key to acknowledge >' ),
    repeat,
    [! beep, delay( 0.25 ) !],
    keyb_peek( A, _ ),
    get0( A ), !.

```

LIST OF REFERENCES

Clark, S. M., "Hydrographic Survey Planning," notes on briefing given to prospective Oceanographic Unit commanding officers, U. S. Naval Oceanographic Office, Stennis Space Center, Mississippi, 13 December 1988.

International Hydrographic Organization (IHO) Special Publication No. 44, *IHO Standards for Hydrographic Surveys*, 3rd ed., International Hydrographic Bureau, 1987.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, v. 220, no. 4598, pp. 671-680, 13 May 1983.

Laurilla, S. H., *Electronic Surveying and Navigation*, pp. 108-117, John Wiley & Sons, Inc., 1976.

Meridian Ocean Systems, *QUILS II Operator's Manual*, p. 57, 1989.

Plastock, R. A., and Kalley, G., *Theory and Problems of Computer Graphics*, pp. 99-104, McGraw-Hill, Inc., 1986.

Rogers, D. F., and Adams, J. A., *Mathematical Elements for Computer Graphics*, 2nd ed., pp. 61-100, McGraw-Hill, Inc., 1990.

Rowe, N. C., *Artificial Intelligence Through Prolog*, pp. 191-247, Prentice-Hall, Inc., 1988.

Umbach, M. J., *Hydrographic Manual*, 4th ed., National Oceanic and Atmospheric Administration, 1976.

Wells, W., and Hart, W. W., *Plane Geometry*, D. C. Heath & Company, 1915.

INITIAL DISTRIBUTION LIST

- | | | |
|----|--|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, VA 22304 | 2 |
| 2. | Library, Code 52
Naval Postgraduate School
Monterey, CA 93943-5000 | 2 |
| 3. | Chairman, Code OC/Co
Department of Oceanography
Naval Postgraduate School
Monterey, CA 93943-5000 | 1 |
| 4. | Neil C. Rowe, Code CS/Rp
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5000 | 2 |
| 5. | Everett Carter, Code OC/Cr
Department of Oceanography
Naval Postgraduate School
Monterey, CA 93943-5000 | 1 |
| 6. | Arnold F. Steed, Code HSA
Naval Oceanographic Office
Stennis Space Center
MS 39522-5001 | 1 |
| 7. | Director of Naval Oceanography Division
Naval Observatory
34th and Massachusetts Avenue NW
Washington, DC 20390 | 1 |
| 8. | Commander
Naval Oceanography Command
Stennis Space Center
MS 39529-5000 | 1 |
| 9. | Commanding Officer
Naval Oceanographic Office
Stennis Space Center
MS 39522-5001 | 1 |

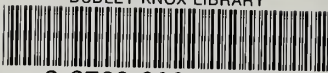
10. Commanding Officer 1
Naval Oceanographic and Atmospheric
Research Laboratory
Stennis Space Center
MS 39529-5004
11. Chairman 1
Oceanography Department
U. S. Naval Academy
Annapolis, MD 21402
12. Chief of Naval Research (Code 420) 1
800 N. Quincy Street
Arlington, VA 22217

Thesis
S67863 Steed
c.1 A heuristic search
method of selecting
range-range sites for
hydrographic surveys.

Thesis
S67863 Steed
c.1 A heuristic search
method of selecting
range-range sites for
hydrographic surveys.



DUDLEY KNOX LIBRARY



3 2768 00036316 2